Kerneld mini-HOWTO

Version 1.7 19 juillet 1997

(Adaptation française par Alexandre Devaure *adevaure@mail.dotcom.fr* , 14 janvier 1999).

Contents

1 Introduction

Ce document explique comment utiliser la fonction `kerneld` avec les noyaux Linux. Il décrit :

- ce qu'est `kerneld` ;
- pourquoi l'utiliser ;
- comment avoir les outils nécessaires ;
- comment les configurer ;
- comment faire fonctionner `kerneld` avec des modules qu'il ne connaît pas ;
- comment espionner `kerneld` (peut s'avérer très utile lors de la mise au point) ;
- les utilisations spéciales de `kerneld` ;
- les problèmes courants et les dysfonctionnements.

La dernière version de ce document peut être trouvée à l'adresse *http://eolicom.olicom.dk/˜storner/kerneld-mini-HOWTO.html* . Entre les versions du mini-HOWTO, vous pouvez trouver des mises à jour sur ma liste non triée des modifications à *http://eolicom.olicom.dk/˜storner/kern.html*

La dernière version française se trouve à l'adresse *http://www.freenix.fr/linux/HOWTO/mini/* .

2 Contributeurs

Si vous découvrez dans ce document des choses fausses, envoyez-moi un mot à ce sujet. Les personnes suivantes ont contribué à ce mini-HOWTO sur certains points :

- Bjorn Ekwall *bjorn@blox.se*
- Ben Gaillart *bgalliac@luc.edu*
- Cedric Tefft *cedric@earthling.net*
- Brian Miller *bmiller@netspace.net.au*
- James C. Tsiao *jtsiao@madoka.jpl.nasa.gov*

J'apprécierai les encouragements et les suggestions des lecteurs de ce mini-HOWTO.

3 Qu'est-ce que `kerneld` ?

`kerneld` est lié à une fonctionnalité introduite lors du développement des noyaux de la série 1.3 par Bjorn Ekwall. Il perdure avec les noyaux 2.0 et 2.1. Il permet aux modules du noyau (c'est-à-dire les pilotes de périphériques, de réseaux, les systèmes de fichiers...) d'être chargés automatiquement en fonction des besoins, sans utilisation manuelle des commandes `modprobe` ou `insmod`.

Des aspects plus amusants, bien que ceux-ci ne soient pas (encore ?) intégrés dans le noyau standard :

- on peut configurer kerneld pour qu'il exécute un programme utilisateur à la place de l'économiseur d'écran, ce qui vous permet d'utiliser n'importe quel programme.
- dans le même genre que l'économiseur d'écran, vous pouvez aussi changer le traditionel “beep” en quelque chose de complètement différent...

`kerneld` est composé de deux entités séparées :

- gestion dans le noyau de Linux afin d'envoyer des requêtes au démon afin de savoir si un module doit être utilisé pour certaines tâches ;
- un démon au niveau utilisateur qui peut montrer quels modules doivent être chargés pour accomplir la requète du noyau.

Ces deux parties doivent fonctionner pour que `kerneld` soit opérationnel. Le fait qu'une des deux soit initialisée ne suffit pas.

4 Pourquoi est-ce que je veux l'utiliser ?

Il y a de bonnes raisons pour utiliser `kerneld`. Voici les miennes. D'autres peuvent l'utiliser pour d'autres raisons.

- Si vous devez construire des noyaux pour de nombreux systèmes qui diffèrent peu (par exemple, une marque différente de carte réseau), alors vous pouvez construire un seul noyau et des modules, à la place d'avoir à construire un noyau par système.

- Les modules sont plus faciles à tester pour les développeurs : il ne faut pas relancer le système pour charger et enlever le pilote. Ceci s'applique pour tous les modules et non juste pour ceux qui sont montés par `kerneld`.
- Il réduit l'usage de la mémoire du noyau, ce qui donne plus de mémoire pour les applications. La mémoire utilisée par le noyau n'est jamais "swappée" sur disque, donc si vous avez 100Ko de pilotes non utilisés compilés dans le noyau, ils occasionnent simplement une perte de RAM.
- Certaines choses que j'utilise, le pilote `ftape`, par exemple ou `iBCS`, ne sont valables que sous forme de modules. Mais je ne veux pas m'embêter avec leur chargement et leur déchargement à chaque fois que j'en ai besoin.
- Les personnes qui font des distributions Linux ne veulent pas construire 284 images de boot différentes, chaque utilisateur charge les pilotes dont il a besoin pour sa configuration. C'est la méthode retenue par la RedHat 4.0 dans son installation.

Bien sûr, il y a aussi des raisons pour que vous ne vouliez pas l'utiliser : vous préfèreriez avoir juste un fichier image de votre noyau avec tous vos pilotes à l'intérieur. Dans ce cas, vous lisez le mauvais document.

5 Où puis-je trouver les outils nécessaires ?

Le support dans le noyau de Linux a été introduit avec Linux 1.3.57. Si vous avez une version plus ancienne, vous devrez la mettre à jour si vous voulez qu'il supporte `kerneld`. Tous les sites ftp majeurs de Linux offrent les sources du noyau. Je recommande que vous le mettiez à jour avec la dernière version 2.0 (actuellement la 2.0.36) :

- *ftp://sunsite.unc.edu/pub/Linux/kernel/v2.0/linux-2.0.36.tar.gz*
- *ftp://tsx-11.mit.edu/pub/linux/sources/system/v2.0/linux-2.0.36.tar.gz*
- *ftp://ftp.funet.fi/pub/Linux/PEOPLE/Linus/v2.0/linux-2.0.36.tar.gz*

Pour les utilisateurs français, il vaut mieux utiliser le miroir francais *ftp://ftp.lip6.fr/pub2/linux/kernel/sources/v2.0/linux-2.0.36.tar.gz*

Le démon en mode utilisateur a été introduit avec le paquetage `modules-1.2.8` et avec le nouveau paquetage `modules-2.0`. Ils sont normalement trouvables à la même place que les sources des noyaux mais les sites officiels sont :

- *ftp://sunsite.unc.edu/pub/Linux/kernel/v2.0/modules-2.0.0.tar.gz*
- *ftp://tsx-11.mit.edu/pub/linux/sources/sbin/modules-2.0.0.tar.gz*
- *ftp://ftp.funet.fi/pub/Linux/tools/modules-2.0.0.tar.gz*

Pour les utilisateurs français : *ftp://ftp.lip6.fr/pub2/linux/kernel/sources/v2.0/modules-2.0.0.tar.gz*

AVERTISSEMENT : si vous voulez essayer de charger des modules avec les derniers noyaux 2.1 (développement), vous devrez utiliser le dernier paquetage `modutils-` (PAS `modules-`). Mais regardez plus bas au sujet des problèmes avec les modules et les noyaux 2.1.

6 Comment le configure-t-on ?

D'abord, ayez les parties nécessaires : un noyau et les derniers `modules-utilities`. Ensuite, vous devez installer les `modules-utilities`. C'est très simple : il faut juste désempaqueter les sources et lancer `make install`. Ceci compile et installe les programmes suivants dans `/sbin` : `genkym`, `insmod`, `lsmod`, `modprobe`, `depmod`, `kerneld`. Je recommande que vous ajoutiez quelques lignes dans les scripts de démarrage pour faire les initialisations nécessaires lors du démarrage de Linux. Ajoutez les lignes suivantes à votre fichier `/etc/rc/rc.d/rc.sysinit` (si vous utilisez la Slackware) ou à `/etc/rc.d/rc.sysinit` (si vous utilisez SysVinit, c'est-à-dire les distributions Debian, RedHat, Caldera) :

```
# Demarrer kerneld - ceci doit arriver tres tot dans le
# processus de demarrage, certainement AVANT que vous lanciez
# fsck sur les systèmes de fichiers qui ont besoins que les
# pilotes de disque soient chargés automatiquement
if [ -x /sbin/kerneld ]
then
    /sbin/kerneld
fi

# Vos commandes fsck fonctionnent ici
# et votre command mount monte le système de fichiers racine
# en lecture seule.

# Mettez à jour le fichier de dépendance des modules du noyau
# Votre système de fichier racine doit être monté en
# lecture-écriture à partir de maintenant
if [ -x /sbin/depmod ]
then
    /sbin/depmod -a
fi
```

La première partie lance `kerneld` lui-même.

La second appelle `depmod -a` au démarrage. Le programme `depmod` construit une liste de tous les modules disponibles et analyse leurs inter dépendances. Donc il sait si un module a besoin qu'un autre soit chargé avant lui.

NOTE : Les versions récentes de `kerneld` ont une option pour utiliser la librairie GNU dbm : `libgdbm`. Si vous l'activez quand vous construisez les `modules-utilities`, `kerneld` ne se lancera pas si `lidgdbm` n'est pas disponible, ce qui pourrait être le cas si vous avez `/usr` sur une partition séparée et que vous lanciez `kerneld`avant que `/usr` ne soit montée. La solution recommandée est de déplacer `libgdm` de `/usr/lib` vers `/lib` ou de faire un lien statique de `kerneld`.

Ensuite, défaîtes les sources du noyau, configurez et construisez un noyau à votre convenance. Si vous ne l'avez jamais fait avant, vous devriez lire le fichire README à la racine des sources du noyau. Quand vous lancez `make config` pour configurer le noyau, vous devrez faire attention à des questions qui apparaissent au début :

```
Enable loadable module support (CONFIG_MODULES) [Y/n/?] Y
```

Vous devez sélectioner la gestion des modules chargeables, sinon, il n'y aura pas de modules à charger pour `kerneld`. Répondez seulement oui (Y).

```
Kernel daemon support (CONFIG_KERNELD) [Y/n/?] Y
```

Ceci est aussi nécessaire. Ensuite, de nombreuses choses peuvent être mises sous forme de modules. Vous verrez des questions du genre :

```
Normal floppy disk support (CONFIG_BLK_DEV_FD) [M/n/y/?]
```

où vous pouvez répondre M pour Module. Généralement, seuls les pilotes nécessaires lors du démarrage de votre système (le pilote du disque dur, le pilote du système de fichiers racine) doivent être mis dans le noyau ; le reste pouvant être construit sous forme de modules.

Quand vous avez fini avec `make config`, lancez `make dep`, `make clean`, `make zImage` ou `make modules`, `make modules` et `make modules_install`.

Ouf !

La commande `make zImage` crée la nouvelle image du noyau dans le fichier `arch/i386/boot/zImage`. Vous devrez le copier où vous mettez votre image de boot. N'oubliez pas de relancer LILO.

Pour plus d'informations sur la configuration, la construction et l'installation de votre propre noyau, regardez le `Kerneld-HOWTO` posté régulièrement au `comp.os.linux.answers` et disponible sur le site `sunsite.unc.edu` à `/pub/Linux/docs/HOWTO`. La version française est disponible à *www.freenix.fr/linux*

7 Tester kerneld

Maintenant, relancez le système avec le nouveau noyau. Quand le système est prêt, vous pouvez exécuter un `ps ax` et vous devriez voir une ligne pour `kerneld` :

```
PID TTY STAT  TIME COMMAND
59  ?  S     0:01 /sbin/kerneld
```

Une des choses intéressantes de `kerneld` est qu'une fois le noyau et le démon installés, seule une très petite initialisation est nécessaire. Pour commencer, essayez d'utiliser un des pilotes que vous avez construit comme module. J'ai construit le pilote de disquette comme module, donc je peux mettre une disquette DOS dans le lecteur et :

```
osiris:~ $ mdir a:
  Volume in drive A has no label
  Volume Serial Number is 2E2B-1102
  Directory for A:/

  binuti~1 gz       1942 02-14-1996  11:35a binutils-2.6.0.6-2.6.0.7.diff.gz
  libc-5~1 gz      24747 02-14-1996  11:35a libc-5.3.4-5.3.5.diff.gz
          2 file(s)        26689 bytes
```

le pilote de disquette fonctionne : il a été chargé automatiquement par `kerneld` quand j'ai voulu utiliser la disquette.

Pour voir que le module `floppy` est en effet chargé, vous pouvez lancer `/sbin/lsmod` qui listera tous les modules chargés à l'instant :

```
osiris:~ $ /sbin/lsmod
Module:        #pages:  Used by:
floppy            11    0 (autoclean)
```

Le mot "autoclean" signifie que le module sera automatiquement enlevé par `kerneld` quand il n'aura pas été utilisé pendant plus d'une minute. Les 11 pages de mémoire (soit 44ko, une page faisant 4ko) seront donc seulement utilisées quand j'accèderai au lecteur de disquette ; si je n'utilise pas la disquette pendant plus d'une minute, elles seront libérées. Très intêressant si vous êtes à court de mémoire pour vos applications !

8 Comment `kerneld` sait-il quel module charger ?

Bien que `kerneld` connaisse déjà les types les plus communs de modules, il y a des situations dans lesquelles `kerneld` ne sera pas comment satisfaire une requête venant du noyau. C'est le cas avec les pilotes de CD-ROM ou de cartes réseau, où il existe plus d'un module possible susceptible d'être chargé.

Les requêtes que le démon de `kerneld` reçoit du noyau viennent d'un des éléments suivants :

- un pilote de périphérique bloc ;
- un pilote de périphérique caractère ;
- un format binaire ;
- une discipline de ligne tty ;
- un système de fichier ;
- un périphérique réseau ;
- un service réseau (par exemple rarp) ;
- un protocole réseau (par exemple IPX).

`kerneld` détermine quel module doit être chargé regardant le fichier de configuration `/etc/conf.modules`. Il y a deux types d'entrée dans ce fichier : les chemins (où les fichiers des modules sont stockés) et les alias (quel module doit être chargé). Si vous n'avez pas déjà ce fichier, vous devrez le créer en lançant `/sbin/modprobe -c | grep -v '^path' > /etc/conf.modules`

Si vous voulez ajouter encore une autre directive "path" aux chemins par défaut, vous devez inclure aussi tous les chemins par défaut étant donné qu'une directive `path` dans `/etc/conf.modules` remplacera toutes celles que `modprobe` connaît par défaut.

Normalement, vous ne voudrez pas ajouter de `path` par vous-même étant donné que l'ensemble des chemins par défaut prend en compte toutes les configurations normales, je vous le promets !

D'un autre côté, si vous voulez juste ajouter un alias ou une directive d'option, vos nouvelles entrées dans `/etc/conf.modules` seront ajoutées à celles que `modprobe` connaît déjà. Si vous deviez redéfinir un alias ou une option, vos nouvelles entrées dans `/etc/conf.modules` remplaceront celles déjà présentes.

8.1 Les périphériques bloc

Si vous lancez `/sbin/modprobe -c`, vous aurez la liste des modules connus par `kerneld` et à quelles requêtes ils correspondent. Par exemple, la requête qui termine le chargement du gestionnaire de disquettes correspond au périphérique bloc dont le numéro majeur est 2 :

```
osiris:~ $ /sbin/modprobe -c | grep floppy
alias block-major-2 floppy
```

Pourquoi `block-major-2` ? Parce que les lecteurs de disquettes `/dev/fd*` utilisent un numéro majeur égal à 2 et sont de type bloc :

```
osiris:~ $ ls -l /dev/fd0 /dev/fd1
brw-rw-rw-   1 root     root       2,   0 Mar  3  1995 /dev/fd0
brw-r--r--   1 root     root       2,   1 Mar  3  1995 /dev/fd1
```

8.2 Les périphériques caractères

Les périphériques de type caractère sont utilisés de la même manière. Par exemple, le lecteur de bande correspond au numéro majeur 27 :

```
osiris:~ $ ls -lL /dev/ftape
crw-rw----   1 root     disk      27,   0 Jul 18  1994 /dev/ftape
```

Toutefois, `kerneld` ne le connaît pas par défaut : il n'est pas listé dans le résultat de `/sbin/modprobe -c`.

Donc, pour configurer `kerneld` de manière à charger le gestionnaire `ftape`, je dois ajouter une ligne au fichier de configuration `/etc/conf.modules` :

```
alias char-major-27 ftape
```

8.3 Les périphériques réseau

Vous pouvez aussi utiliser le nom du périphérique à la place de `char-major-xxx` ou `block-major-yyy`. Ceci est particulièrement utilisé pour les gestionnaires réseaux. Par exemple, un pilote pour une carte réseau ne2000 utilisée comme `eth0` pourrait être chargé avec :

```
alias eth0 ne
```

Si vous devez passer des options au gestionnaire (comme de dire au module quelle IRQ la carte réseau utilise), vous ajoutez une ligne `options` :

```
options ne irq=5
```

Ainsi `kerneld` lancera le gestionnaire NE2000 avec la commande :

```
/sbin/modprobe ne irq=5
```

Bien sûr, les options disponibles sont spécifiques aux modules que vous chargez.

8.4 Les formats binaires

Les formats binaires sont gérés de la même façon. A chaque fois que vous essayez de lancer un programme que le noyau ne sait pas comment exécuter, `kerneld` lance une requête pour `binfmt-xxx`, ou `xxx` est le nombre déterminé à partir des tous premiers octets de l'exécutable. Donc la configuration de `kerneld` pour la gestion du chargement du module binfmt_aout pour les exécutable ZMAGIC (a.out) est :

```
alias binfmt-267 binfmt_aout
```

vu que le nombre magique pour les fichiers ZMAGIC est 267 (voir `/etc/magic`). Si vous regardez `/etc/magic`, vous verrez le nombre 0413, ceci parce que ce fichier utilise des nombres octaux alors que `kerneld` utilise des décimaux (413 en octal correspond à 267 en décimal). Il y a en réalité trois variantes des exécutables a.out peu différentes (NMAGIC, QMAGIC et ZMAGIC). Pour un support total du format a.out, vous devez avoir :

```
alias binfmt-264 binfmt_aout  # pure executable (NMAGIC)
alias binfmt-267 binfmt_aout  # demand-paged executable (ZMAGIC)
alias binfmt-204 binfmt_aout  # demand-paged executable (QMAGIC)
```

Les formats binaires a.out, Jave et iBCS sont reconnus automatiquement par `kerneld` sans la moindre configuration.

8.5 Les disciplines de ligne (slip, cslip et ppp)

Les disciplines de lignes sont demandées avec *tyy-ldisc-x* où *x* est généralement 1 (pour SLIP) ou 3 (pour PPP). Ces deux sont reconnus automatiquement par `kerneld`.

Concernant PPP, si vous voulez que `kerneld` charge le module de compression de données pour PPP `bsd_comp`, vous devez ajouter les deux lignes suivantes au fichier `/etc/conf.modules` :

```
alias tty-ldisc-3 bsd_comp
alias ppp0 bsd_comp
```

8.6 Les familles de protocoles réseau (IPX, AppleTalk, AX.25)

Certains protocoles réseau peuvent être aussi chargés sous la forme de modules. Le noyau demande à `kerneld` une famille de protocole (par exemple IPX) avec une requête pour *net-pf-X* où *X* est un nombre indiquant la famille voulue. Par exemple, *netpf-3* correspond à AX.25, *net-pf-4* à IPX et *net-pf-5* à AppleTalk. (Ces nombres sont déterminés par les macros `AF_AX25`, `AF_IPX` etc., que l'on trouve dans le fichier source `include/linux/socket.h`. Donc, pour charger automatiquement le module IPX, vous devrez ajouter une entrée dans `/etc/conf.modules` :

```
alias net-pf-4 ipx
```

Consultez également la section traitant des problèmes courants pour éviter des messages d'avertissment lors de l'amorçage relatifs à des familles de protocoles indéfinies.

8.7 Les systèmes de fichiers

Les requêtes soumises à `kerneld` pour les systèmes de fichiers sont simplement constituées par le type du système de fichiers. Un usage courant est de charger le module *isofs* pour les systèmes de fichiers des CD-ROM, c'est-à-dire les systèmes de fichiers de type *iso9660* :

```
alias iso9660 isofs
```

9 Périphériques demandant une configuration spéciale

Certains périphériques demandent un peu plus de configuration que le simple alias d'un périphérique et d'un module.

- les périphériques de type caractère de numéro majeur 10 : divers périphériques ;
- les périphériques SCSI :
- les périphériques qui demandent une initialisation spéciale.

9.1 char-major-10 : souris, watchdogs, et random

Les périphériques sont habituellement identifiés par leur nombre majeur, par exemple 27 pour `ftape`. Toutefois, si vous regardez les entrées de `/dev` pour le nombre majeur 10, vous verrez un certain nombre de périphériques très différents. Parmi ceux-ci :

- des souris de toutes sortes (souris bus, PS/2,...) ;
- les chiens de garde (watchdog) ;
- le périphérique noyau *random* ;
- l'interface APM (Advanced Power Management).

De façon évidente, ces périphériques sont contrôlés par différents modules et non un seul. Pour cela, `kerneld` utilise le nombre majeur et le nombre mineur :

```
alias char-major-10-1 psaux      # For PS/2 mouse
alias char-major-10-130 wdt      # For WDT watchdog
```

Vous avez besoin d'un version du noyau 1.3.82 ou supérieure pour l'utiliser. Les versions plus anciennes ne passaient pas le nombre mineur à `kerneld`, ce qui ne permettait pas à `kerneld` de savoir quel module il fallait charger.

9.2 Charger les gestionnaires SCSI : l'entrée *scsi_hostadapter*

Les gestionnaires de périphériques SCSI sont constitués d'un adaptateur pour la carte SCSI (par exemple pour une Adaptec 1542) et d'un gestionnaire pour le type de périphérique SCSI que vous utilisez, comme un disque dur, un lecteur de CD-ROM ou un lecteur de cartouche. Tous peuvent être chargés sous forme de modules. Cependant, lorsque vous voulez accéder à un lecteur de CD-ROM connecté à une carte Adaptec, le noyau et `kerneld` savent seulement qu'il faut charger le module *sr_mod* pour gérer le CD-ROM SCSI, mais ils ignorent à quel contrôleur SCSI il est connecté, donc quel module charger pour gérer le contrôleur SCSI.

Pour résoudre cela, vous pouvez ajouter une entrée pour le module du contrôleur SCSI au fichier `/etc/conf.modules` qui indiquera à `kerneld` quel module charger parmi toutes les possibilités :

```
alias scd0 sr_mod               # sr_mod pour SCSI CD-ROM's ...
alias scsi_hostadapter aha1542  # ... doit utiliser le pilote
                                # Adaptec 1542
```

Cela ne fonctionne que pour un noyau de version 1.3.82 ou supérieure.

Cela marche si vous n'avez qu'une carte SCSI, sinon, c'est un peu plus difficile. En général, vous ne pouvez pas avoir `kerneld` qui charge le pilote d'une carte SCSI si le gestionnaire d'un autre contrôleur est déjà installé. Vous devez soit construire un noyau avec les deux gestionnaires (ils ne sont plus sous forme de modules) soit les charger manuellement.

Il y a une possibilité pour que `kerneld`charge plusieurs gestionnaires SCSI. James Tsiao a eu cette idée : vous pouvez avoir `kerneld` qui charge le second controleur SCSI en mettant la dépendance dans le fichier `modules.dep` à la main. Vous avez juste besoin d'une entrée comme :

```
/lib/modules/2.0.30/scsi/st.o: /lib/modules/2.0.30/scsi/aha1542.o
```

Pour que `kerneld`charge le module *aha1542.o* avant qu'il charge *st.o*. Ma machine à la maison est configurée exactement comme au-dessus et fonctionne très bien pour tous les périphérique de mon second contrôleur SCSI, incluant lecteurs de cartouche, CD-ROM et des périphériques SCSI génériques. L'inconvéniant est que `depmod -a` ne peut pas détecter ces dépendances. Donc, l'utilisateur doit les ajouter à la main et ne pas lancer `depmod -a` au démarrage. Une fois configuré, `kerneld` chargera automatiquement *aha1542.o* comme il faut.

Vous devez être conscient que cette technique ne marche que si vous avez différents types de périphériques sur deux contrôleurs. Par exemple les disques durs sur un contrôleur et les lecteurs de CD-ROM, de cartouches et les périphériques génériques sur l'autres.

9.3 Quand charger un module n'est pas suffisant : l'entrée `post-install`

Parfois, charger un module n'est pas suffisant pour qu'il fonctionne correctement. Par exemple, si vous avez compilé le pilote de votre carte son en tant que module, il est souvent pratique de le régler pour un certain volume sonore. Le seul problème, c'est que cette initialisation disparaît lors du chargement suivant du module. Voici un truc de Ben Galliart *bgailla@luc.edu* :

Il faut installer le paquetage `setmix-0.1` (*ftp://sunsite.unc.edu/pub/Linux/apps/sound/mixers/setmix-0.1.tar.gz*)

et ensuite ajouter les lignes suivantes au fichier `/etc/conf.modules` :

```
post-install sound /usr/local/bin/setmix -f /etc/volume.conf
```

Ainsi `kerneld` exécute la commande indiquée par l'entrée `post-install sound` après que le module son ait été chargé. Donc, le module son est configuré par la commande `/usr/local/bin/setmix -f /etc/volume.conf`.

Cela peut s'avérer très utile pour d'autres modules, par exemple le module *lp* peut être configuré par le programme `tunelp` en ajoutant :

```
post-install lp tunelp <options>
```

Pour que `kerneld`reconnaisse ces options, vous devez avoir une version 1.3.69 de `kerneld` ou supérieure.

Note : une version précédente de ce mini-HOWTO mentionne une option `pre-remove` qui peut être utilisée pour excécuter une commande juste avant que `kerneld` ne décharge un module. Toutefois, cela n'a jamais marché et son utilisation est déconseillée. Heureusement, cette options disparaitra dans une future version de `kerneld`. L'ensemble des opérations d'initialisation des modules est en cours de modification en ce moment, et peut différer sur votre système au moment où vous lirez ceci.

10 Espionner kerneld

Si vous avez tout essayé et que vous ne comprenez pas ce que le noyau demande à `kerneld`, il y a une solution pour voir les requêtes que reçoit `kerneld` et par conséquent comprendre ce qu'il faut mettre dans `/etc/conf.modules`. Pour cela, il faut utiliser l'utilitaire `kdstat`.

Ce petit programme est livré avec le paquetage `modules`, mais il n'est ni compilé, ni installé par défaut. Pour le compiler :

```
cd /usr/src/modules-2.0.0/kerneld
make kdstat
```

Ensuite, pour que `kerneld` affiche les informations sur ce qu'il est en train de faire, il faut lancer :

```
kdstat debug
```

et `kerneld` commencera à envoyer des messages à la console sur son activité. Si vous essayez de lancer la commande que vous voulez utiliser, vous verrez les requêtes adressées à `kerneld`. Elles peuvent être copiées dans le fichier `/etc/conf.modules` et mises en alias du module demandé pour réaliser la tâche.

Pour arrêter le debuggage, lancez :

```
/sbin/kdstat nodebug
```

11 Utilisations spéciales de kerneld

Je savais bien que vous me demanderiez comment configurer le module d'économiseur d'écran...

Le répertoire `kerneld/GOODIES` dans le paquetage `modules` a un certain nombre de patches noyau pour la gestion de l'économiseur d'écran ainsi que le beep de la console par `kerneld`. Ils ne font pas partie du noyau officiel. Vous devrez donc installer les patches noyau et le recompiler.

Pour installer un patch, utilisez la commande “patch” :

```
cd /usr/src/linux
patch -s -p1 </usr/src/modules-2.0.0/kerneld/GOODIES/blanker_patch
```

Ensuite recompilez et installez le nouveau noyau.

Quand il sera temps de lancer l'économiseur d'écran, `kerneld` exécutera la commande `/sbin/screenblanker` (il peut s'agir d'un script shell qui lance votre économiseur d'écran favorixt.

Quand le noyau veut l'arrêter, il envoie un signal `SIGQUIT` au processus exécutant `/sbin/screenblanker`. Votre script shell ou économiseur d'écran doit le capter et se terminer. Pensez à restaurer l'écran dans le mode texte initial !

12 Problèmes courants

12.1 Pourquoi est-ce que j'ai des messages “Cannot locate module for net-pf-X” quand j'excécute `ifconfig` ?

Le code du noyau a été modifié pour permettre le chargement des familles de protocoles réseau (comme IPX, AX25 et AppelTalk) comme modules vers la version 1.3.80. Cela a eu pour effet d'ajouter une nouvelle requête pour `kerneld`, *net-pf-X*, où *X* est un nombre identifiant le protocole (voir le fichier `/usr/src/linux/include/linux/socket.h` pour la signification de ces nombres).

Malheureusement, `ifconfig` envoie ces messages, donc un bon nombre de personnes recoivent ces messages lors le système se lance et qu'il exécute `ifconfig` pour initialiser le périphérique `loopback`. Ces messages sont sans danger et vous pouvez les retirer en ajoutant les lignes suivantes :

```
alias net-pf-3 off  # oubliez AX.25
alias net-pf-4 off  # oubliez IPX
aliad net-pf-5 off  # oubliez AppleTalk
```

au fichier `/etc/conf.modules`. Biensûr, si vous utilisez IPX comme module, n'ajoutez pas la ligne qui retire IPX.

12.2 Après voir lancer `kerneld`, mon système ralentit quand j'active ma connexion PPP

Il y a bon nombre de messages à ce sujet. Il semble qu'il y ait une malheureuse interaction entre `kerneld` et le script `tkPPP` qui est utilisé sur certains systèmes pour configurer et surveiller la connexion PPP. Le script exécute apparemment des boucles quand il lance `ifconfig`. Celui-ci déclenche `kerneld` pour rechercher les modules *net-pf-X* (voir ci-dessous), ce qui provoque une surcharge du système et l'envoi possible de messages `“Cannot locate module for net-pf-X”`. Il n'y a pas d'autres solutions que de ne pas utiliser `tkPPP` ou de changer sa façon de surveiller la connexion.

12.3 `kerneld`ne charge pas mon gestionnaire SCSI

Ajoutez une entrée pour la carte SCSI au fichier `/etc/conf.modules`. Regardez la description de l'entrée `scsi_hostadapter` plus haut.

12.4 `modprobe` se plaint que `gcc2_compiled` n'est pas défini

Ceci est une erreur dans `module-utilities` qui ne se voit qu'avec `binutils 2.6.0.9` ou supérieur et elle est aussi documentée dans les notes de mises à jour du paquetage `binutils`. Lisez-le donc ou mettez à jour le paquetage des modules par un qui corrige ce problème, par exemple le `modules-2.0.0`.

12.5 Le volume de ma carte son n'est pas initialisé etc.

Les options de configuration d'un modules sont stockées dans le module lui-même quand il est chargé. Donc, quand `kerneld` décharge un module, la configuration que vous aviez faite est perdue et la prochaine fois que le module sera chargé, il héritera de la configuration par défaut.

Vous pouvez indiquer à `kerneld` de configurer un module en exécutant un programme après son chargement automatique. Voir la section sur l'entrée `post-install`.

12.6 DOSEMU a besoin de modules, comment `kerneld` peut-il les charger ?

Vous ne pouvez pas. Aucune des versions de `dosemu` (officielles ou de développement) ne gèrent le chargement des modules à travers `kerneld`. Cependant, if vous avez un noyau 2.0.26 ou supérieur, vous n'avez pas besoin de modules dosemu particuliers. Installez juste dosemu 0.66.1.

12.7 Pourquoi ai-je des messages "Ouch, kerneld time out, message failed" ?

Quand le noyau envoit une requête à `kerneld`, il s'attend à recevoir un acquittement dans un délai d'une seconde. Si `kerneld` n'envoie pas cet acquittement, ce message est diffusé. La requête est retransmise et peut éventuellement réussir

Cela arrive couramment sur des systèmes lourdement chargés. `kerneld` étant un processus en mode utilisateur, il est ordonnancé comme tout processus du système. Sous de fortes charges, il peut ne pas s'exécuter pour envoyer l'acquittement avant l'expiration du délai.

Si cela se produit quand la charge est faible, essayez de redémarrer `kerneld`. Tuez le processus `kerneld` et redémarrez-le avec la commande `/usr/sbin/kerneld`. Si le problème persiste, vous devrez envoyer un message d'erreur à *linux-kernel@vger.rutgers.edu* mais, **s'il vous plaît** soyez sûr que votre version du noyau et de `kerneld` soient à jour avant d'envoyer un message sur ce problème.

12.8 `mount` n'attend pas que `kerneld` charge le module du système de fichier

Il existe un certain nombre de messages sur le fait que la commande `mount(8)` n'attende pas que `kerneld` ait chargé le module du système de fichiers. `lsmod` montre que `kerneld` a chargé le module et si vous répétez la commande `mount` immédiatement, le montage sera réussi. Cela semble être une erreur dans le paquetage `modules` version 1.3.69f qui affecte des utilisateurs de Debian (elle peut être corrigée en installant la dernière version de ce paquetage).

12.9 `kerneld` n'arrive pas à charger le module *ncpfs*

Vous devez compiler les utilitaires `ncpfs` avec l'option `-DHAVE_KERNELD`. Voir le fichier `Makefile` de `ncpfs`.

12.10 `kerneld` n'arrive pas à charger le module *smbfs*

Vous utilisez une vieille version des utilitaires `smbmount`. Prenez la dernière version (0.10 ou supérieure) à *ftp://tsx-11.mit.edu/pub/linux/filesystems/smbfs/* .

12.11 J'ai tout recompilé sous forme de modules et maintenant, mon système ne peut plus démarrer : `kerneld` n'arrive pas à charger le module du système de fichier racine.

Vous ne pouvez pas **tout** mettre sous forme de modules : le noyau doit avoir assez de gestionnaires pour monter votre système de fichiers racine et exécuter les programmes nécessaires au démarrage de `kerneld`. Donc vous ne pouvez pas mettre sous forme de modules :

- le gestionnaire de votre disque dur où réside votre système de fichiers racine ;
- le gestionnaire du système de fichiers racine ;
- le chargeur du format de binaire pour `init`, `kerneld` et d'autres prgrammes.

En fait ce n'est pas vrai. Les dernières version 1.3.x et toutes les 2.x du noyau, supportent l'utilisation d'un disque ram qui est chargé par `lilo` ou `loadlin` et il est possible de charger des modules de ce "disque" très tôt dans le processus de démarrage. La marche à suivre est décrite dans le fichier `Documentation/initrd.txt` dans l'arborescence des sources du noyau.

12.12 `kerneld`ne se lance pas lors de l'amorçage de la machine : il veut *libgdbm*

Les nouvelles versions de `kerneld` ont besoin de la librairie GNU dbm, *libgdbm.so* pour fonctionner. La plupart des installations ont ce fichier dans `/usr/lib` mais vous avez probablement lancé `kerneld` avant que le système de fichiers de `/usr` ne soit monté. Un des symptomes de ceci est que `kerneld` ne marche pas lors du démarrage du système et de l'exécution des script rc, mais fonctionne parfaitement si vous le lancez à la main après. La solution est soit de déplacer le lancement de `kerneld` après que `/usr` ne soit monté, soit de mettre la librairie *gdbm* dans le système de fichiers racine (par exemple dans `/lib`).

12.13 J'ai "Cannot load module xxx" mais j'ai reconfiguré mon noyau sans la gestion de xxx !

L'installation de la Slackware (et peut-être d'autres) crée un fichier `/etc/rc.d/rc.modules` par défaut qui fait un `modprobe` explicite sur une grande variété de modules. Quels modules exactement sont "modprobés" ?, cela dépend de la configuration initiale du noyau. Vous avez probablement reconfiguré votre noyau pour enlever un ou plusieurs modules qui est modprobé dans `rc.modules`, d'où les messages d'erreur. Mettez à jour votre fichier `rc.modules` en commentant tout module que vous n'utilisez plus, ou enlevez entièrement ce fichier et laissez `kerneld` charger les modules quand on en a besoin.

12.14 J'ai recompilé mon noyau et les modules et j'ai toujours des messages sur des symboles non résolus au démarrage

Vous avez probablement reconfiguré et recompilé votre noyau et exclu des modules. Vous avez d'anciens modules que vous n'utilisez pas dans le répertoire `/lib/modules`. La solution la plus simple est d'effacer le répertoire `/lib/modules/x.y.z` et de retaper `make modules_install` depuis le répertoire des sources du noyau. Notez que ce problème arrive seulement quand vous reconfigurez le noyau sans changer de version. Si vous voyez cette erreur quand vous passer à une nouvelle version du noyau, vous avez un autre problème.

12.15 J'ai installé Linux 2.1 et aucun module ne se charge

Linux 2.1 est un noyau de développement. Pour cette raison, il se peut que certaines choses ne fonctionnent pas de temps en temps. La façon dont les modules sont manipulés a changé de façon significative. Richard Henderson a la charge du développement du noyau des modules.

En bref, si vous voulez utiliser les modules avec un noyau 2.1, vous devez :

- lire le fichier `Documentation/Changes` et voir quels paquetages doivent être mis à jour sur votre système ;
- utiliser le dernier paquetage `modutils`, disponible sur *ftp://ftp.redhat.com/pub/alphabits/* ou sur le site mirroir *ftp://tsx-11.mit.edu/pub/linux/packages/alphabits/*

Je recommande le noyau 2.1.29, si vous voulez utiliser les modules avec un noyau 2.1.

12.16 Que dire d'un réseau utilisant la ligne téléphonique ?

`kerneld` peut à l'origine gérer l'établissement de connexions réseau à travers le réseau téléphonique à la demande : essayer d'envoyer des paquets à un réseau sans être connecté, peut entraîner `kerneld` à lancer le script `/sbin/request_route` pour initialiser une connexion PPP ou SLIP.

Il s'est avéré que c'était une mauvaise idée. Alan Cox, bien connu pour ses travaux sur le réseau dans Linux a écrit sur la liste de diffusion linux-kernel que :

"Le truc request-route est obsolète, cassé et non requis... Il est aussi enlevé des versions 2.1.x."

A la place d'utiliser le script `request-route` et `kerneld`, je vous encourage vivement à installer le paquetage `diald` d'Eric Schenk, disponible à l'url *http://www.dna.lth.se/˜erics/diald.html*

13 Copyright

être données sous certaines conditions. Contactez le coordonnateur des HowTo Linux à l'adresse donnée un peu plus bas.

En résumé, nous souhaitons promouvoir la diffusion de ces informations à travers un maximum de moyens de communication. Toutefois, nous souhaitions conserver un copyright sur les documents HowTo et nous souhaitons être avertis de leur redistribution.

Si vous avez des questions, vous pouvez contacter Greg Hankins, le coordonnateur des HowTo Linux par courrier électronique à *gregh@sunsite.unc.edu*