Renforcer une station Linux face à un risque d'arrêt intempestif*

Le but de cet article est de fournir des éléments pour pouvoir configurer une machine Linux afin qu'elle accepte de fonctionner dans un environnement hostile, sujet à des coupures plus ou moins fréquentes d'alimentation électrique ou à des interventions humaines d'une pertinence discutable. Nous allons donc essayer de mettre en place un système où les données figées seront regroupées sur des partitions en lecture seule, et où les partitions temporaires seront reformatées à chaque démarrage.

L'une des premières choses que l'on apprend lors de l'installation d'un système Linux, est qu'il ne faut jamais éteindre brutalement la machine, mais qu'il faut au contraire toujours passer par une phase de *shutdown* qui permet un arrêt propre. En fait, j'ai observé que les nouveaux utilisateurs actuels commencent à prendre spontanément cette habitude, puisque habitués à cliquer sur un bouton nommé *Démarrer* pour éteindre leur ordinateur, ils n'ont plus le réflexe de couper sauvagement l'alimentation électrique comme le faisait les premiers transfuges du monde Dos.

Il est donc clair qu'il ne faut jamais presser inconsidérément le bouton d'arrêt d'une station Linux. C'est **mal** ! Cela dit, il est important de comprendre pourquoi une telle règle s'impose, car il existe des situations où l'on est obligé de la contourner. Je peux citer le cas d'un ordinateur embarqué en voiture, branché directement sur le contact du démarreur, pour une expérimentation de système de radiolocalisation. La consommation du système ne permettait pas de le laisser fonctionner en permanence sur la batterie, et il était plutôt difficile de demander aux multiples utilisateurs du véhicule (service technique) de faire un *shutdown* avant de couper le contact ! D'autant que le terminal visible n'était muni que de quatre touches correspondant aux flèches de direction

Je mentionnerai également un système de supervision dont des postes de contrôle sont installés dans des locaux techniques (chaufferie) où non seulement la distribution électrique est parfois interrompue intempestivement, mais où un personnel de maintenance plus ou moins qualifié en informatique peut être amené à intervenir sur la machine (ajout ou retrait de cartes d'acquisition, connexion sur des automates industriels, etc.) J'imagine que le même genre de problème doit se poser dans les cybercafés, sur les bornes publiques d'information ou dans les salles d'enseignement en libre accès.

Dans de telles conditions, l'idéal est de disposer d'une configuration la plus robuste possible, acceptant sans broncher les arrêts brutaux de l'ordinateur.

Arrêt d'un système Linux

Pour comprendre dans quelle mesure il est utile de respecter les procédures de *shutdown* sur un système Linux, on peut examiner la liste des opérations auxquelles le système se livre lorsqu'on invoque la commande d'arrêt. On se basera ici sur une distribution Red-Hat 6.1, mais l'essentiel de notre propos reste valable pour les autres systèmes, il suffit de fouiller un peu dans les scripts se trouvant dans `/etc/rc.d` pour retrouver les opérations équivalentes.

Pour arrêter une station, on invoque de préférence `/sbin/init` pour basculer au niveau d'exécution `0` (éteindre l'ordinateur) ou `6` (redémarrer la machine). On peut aussi invoquer directement `/sbin/halt`, `/sbin/reboot` ou `/sbin/shutdown`, mais ces utilitaires invoquent d'abord `/sbin/init` si l'on ne se trouve pas au niveau d'exécution adéquat. On trouve donc dans le répertoire `/etc/rc.d/rc0.d/` un ensemble de scripts qui sont appelés pour terminer les applications en cours d'exécution sur le système.

En premier lieu, les démons sont arrêtés (`gpm`, `httpd`, `nfs`, etc.) ensuite le système interrompt les services réseau (courrier, résolution de noms, interfaces) enfin, les dernières opérations consistent essentiellement en quatre étapes :

- Envoi d'un signal `TERM` à tous les processus restants pour leur demander ‘‘ gentiment ’’ de se terminer. Le système attend quelques secondes.
- Envoi d'un signal `KILL` aux éventuels processus restants, pour les tuer inconditionnellement.
- Tentative de démontage de toutes les partitions, et désactivation du *swap*. La partition racine est alors remontée en lecture seule – elle n'avait pas pu être démontée précédemment puisque le processus `init` la maintient occupée.

---

* Cet article a été publié dans le numéro 20 de Linux Magazine France, en Septembre 2000.

- Finalement, appel de `/sbin/shutdown` (via `/sbin/halt`), qui s'occupera de l'arrêt effectif de la machine (ou de sa réinitialisation si on emploie `/sbin/reboot`).

En fait, l'instant crucial de cette suite d'opération est le troisième point présenté ci-dessus, le démontage des partitions.

Le noyau Linux utilise un système de mémoire tampon pour retarder au maximum les écritures effectives sur le disque. Ainsi lorsque des données sont envoyées au noyau, par l'intermédiaire d'un appel-système `write(2)`, celui-ci les conserve en mémoire vive le plus longtemps possible avant de les transmettre effectivement au contrôleur du disque. Si on modifie à nouveau le bloc de données avant son écriture réelle, les changements n'auront lieu qu'en mémoire vive, sans impliquer le disque physique. Cela permet de gagner beaucoup de temps : par exemple lorsqu'on ajoute successivement des données à la fin d'un fichier, sa longueur est modifiée à maintes reprises. Pourtant, si la mémoire tampon joue bien son rôle, le champ de longueur du fichier n'est écrit qu'à une seule reprise sur le disque.

En contrepartie, il devient obligatoire, avant d'arrêter une station Linux, de demander au noyau de vider totalement sa mémoire tampon. Sur les premiers systèmes Unix, on se contentait d'invoquer l'utilitaire `/sbin/sync`, qui demande au noyau, par l'intermédiaire de l'appel-système `sync(2)`, de réaliser immédiatement toutes les écritures différées en attente. Avec le système de fichiers `ext2`, utilisé pour les partitions Linux, un mécanisme supplémentaire a été ajouté. Un bit d'état est présent sur le disque, indiquant que la partition est dans un état instable. Ce bit est mis à 1 lorsque le noyau monte le système de fichiers en lecture et écriture. Il est remis à 0 lorsque le noyau démonte le système de fichiers car l'état des données sur le disque est alors connu. Si la machine est arrêtée avant d'avoir démonté une partition, cette dernière est donc marquée comme instable, et au redémarrage de la machine, les scripts d'initialisation vont déclencher une vérification du système de fichiers avec `/sbin/fsck`.

La vérification d'une partition complète pouvant être très longue – lorsqu'elle s'étend sur plusieurs gigaoctets – on assiste actuellement au déploiement de nouveaux systèmes de fichiers *journalisés* (`ext3`, `reiserfs`, `xfs`), sur lesquels le noyau tient un journal des blocs de données en cours de modification. Lors du redémarrage après un arrêt brutal, seules les parties du disque dernièrement modifiées auront besoin d'être vérifiées. Cela est surtout utile pour les applications manipulant de grosses bases de données avec des contraintes importantes sur le temps de fonctionnement (nécessitant donc un redémarrage le plus rapide possible après un incident).

Pallier les arrêts intempestifs

Pour essayer de rendre une station Linux plus robuste vis-à-vis des extinctions brutales, il est donc nécessaire d'agir sur les montages et démontages de partitions. Nous devons examiner quelles sont les données manipulées par la machine, et trouver une solution pour les protéger au maximum. En fait, nous observons qu'il existe sur le disque 3 types de fichiers :

- Les fichiers qui ne sont modifiés qu'exceptionnellement : les exécutables (`/bin`, `/usr/bin`), les bibliothèques (`/lib`, `/usr/lib`), les fichiers de configuration du système (`/etc`), les n°uds spéciaux représentant des périphériques (`/dev`). Toute modification de ces éléments peut être considérée comme une opération de maintenance, durant laquelle le système n'est pas dans son état de fonctionnement habituel. On peut donc imposer que ces fichiers soient immuables en situation normale, et se trouvent sur une ou plusieurs partitions montées en lecture seule. Une partition en lecture seule ne risquant pas d'être endommagée par un arrêt brutal, elle ne nécessite pas de vérification lors du démarrage suivant.

- Les fichiers à faible durée de vie. Il s'agit de données temporaires utilisées par les applications durant leur exécution (comme `/tmp`). La caractéristique principale de ces fichiers est qu'ils n'ont pas besoin de continuer à exister lorsque l'application est arrêtée – ni *a fortiori* lorsque la machine est relancée. Nous pouvons donc regrouper toutes ces données sur une partition qui sera reformatée à chaque démarrage de la machine.

- Les fichiers ayant une durée de vie moyennement longue. Il s'agit par exemple des données appartenant aux utilisateurs (`/home`). Ces informations sont caractérisées par le fait qu'elles doivent être parfois modifiées mais qu'elles doivent persister entre deux redémarrages de la machine. Dans certaines situations, il n'y a pas besoin d'employer de telles données. Dans un système embarqué en voiture, dans une machine dédiée à la supervision de processus industriels ou dans un terminal X, ces fichiers n'ont pas de raison d'être. Dans les autres cas, il faudra trouver une solution pour protéger les données.

Nous allons dans un premier temps nous intéresser aux machines n'utilisant que les deux premiers types de fichiers. L'utilisation de données persistant entre deux redémarrages sera abordée en fin d'article.

Pour utiliser un exemple concret, nous allons configurer une station X-Window. Il s'agit d'une installation effectuée à partir d'une distribution Red-Hat 6.1, la machine étant paramétrée pour démarrer automatiquement sous X-Window, en employant `xdm` pour interroger les stations présentes sur le réseau afin de se connecter sur celles qui l'acceptent.

Configuration d'une station

Nous allons utiliser sur notre système 3 partitions :

- La partition racine `/` sera normalement montée en lecture seule. Elle regroupera tous les fichiers persistants du système.

- Une partition `/var` sera formatée à chaque démarrage de la machine, et elle contiendra tous les fichiers temporaires du système. Au besoin, des liens

symboliques seront établis sur la partition racine pour pointer au sein de la partition `/var`. Par exemple le répertoire `/tmp` sera en réalité un lien symbolique vers `/var/tmp`.

- Enfin, le système a besoin d'une partition de *swap*, qui sera reformatée aussi à chaque démarrage par acquit de conscience.

La machine ainsi configurée sera donc une sorte de terminal X intelligent, capable de démarrer et d'être supervisé de manière autonome – à la différence des terminaux X classiques qui ont en général besoin de charger leur propre système d'exploitation depuis un hôte situé sur le même réseau. On pourra éteindre l'ordinateur sans se soucier des procédures d'arrêt usuelles.

Organisation des systèmes de fichiers

Comme nous l'avons évoqué plus haut, nous allons disposer tout d'abord des systèmes de fichiers suivants :

Point	Type	Partition	Particularités
`/`	`ext2`	hdb1	Montage en lecture seule.
`/var`	`ext2`	hdb2	Reformatée à chaque démarrage.
	`swap`	hdb3	Reformatée à chaque démarrage
`/proc`	`proc`		Pseudo système de fichiers géré par le noyau
`/dev/pts`	`devpts`		Pseudo système de fichiers géré par le noyau

Ce tableau nous permet déjà de prévoir comment sera configuré le fichier `/etc/fstab` :

```
/dev/hdb1  /         ext2    ro,defaults 1  1
/dev/hdb2  /var      ext2    defaults    1  2
/dev/hdb2  swap      swap    defaults    0  0
none       /dev/pts  devpts  defaults    0  0
none       /proc     proc    gid=5       0  0
```

Pendant toute la phase de mise au point du système, il sera nécessaire d'employer à plusieurs reprises les commandes :

```
mount / -o ro,remount
```

et

```
mount / -o rw,remount
```

qui servent, en cours d'utilisation, à remonter la partition racine respectivement en lecture seule et en lecture / écriture. En effet, il faudra procéder par essais et erreurs, en tentant de monter la partition en lecture seule, et en laissant le système s'initialiser jusqu'à arriver un message d'erreur, que l'on analysera. Ensuite on remonte la partition en lecture-écriture, et on corrige le défaut. Ceci jusqu'à obtenir un fonctionnement ne présentant plus de problèmes.

Durant cette phase de mise au point, je conseille d'employer une astuce dans `/etc/inittab`, en modifiant la ligne :

```
ca::ctrlaltdel:/sbin/shutdown –t3 –r –now
```

en

```
ca::ctrlaltdel:/sbin/init 1
```

Ainsi il sera possible (après avoir demandé à `init` de relire ce fichier en employant `kill –1 1`) de basculer en mode mono-utilisateur simplement en pressant les touches Contrôle-Alt-Suppr. Attention ! Cette configuration représente une énorme faille de sécurité puisqu'il suffit de presser trois touches pour devenir *root* ; il s'agit simplement d'une modification provisoire, permettant de reprendre facilement le contrôle si le système se bloque.

Fichiers d'initialisation

La distribution Red-Hat emploie par défaut l'utilitaire `mingetty` pour gérer les connexions sur les pseudo terminaux. Malheureusement cette application ne supporte pas d'échouer en tentant de modifier l'appartenance ou les permissions du fichier spécial de périphérique `/dev/tty` qui lui correspond. Comme ce dernier se trouve sur une partition en lecture seule `mingetty` se termine abruptement sans possibilité de contourner cet échec, aussi devrons-nous rechercher un autre utilitaire de connexion. Je conseille d'employer le paquetage `mgetty+sendfax`, qui est au demeurant plus puissant que `mingetty` puisqu'il permet de gérer des connexions par modem. Nous modifierons donc dans `/etc/inittab` les lignes :

```
1:2345:respawn:/sbin/mingetty tty1
```

en

```
1:2345:respawn:/sbin/mgetty –rb tty1
```

Par la même occasion, on transformera le degré de volubilité de `mgetty` en remplaçant la ligne :

```
debug 4
```

se trouvant dans `/etc/mgetty+sendfax/mgetty.config` par

```
debug 0
```

Nous pouvons à présent attaquer la configuration du script principal d'initialisation `/etc/rc.d/rc.sysinit`. Tout d'abord nous ajouterons une ligne pour reformater la partition de *swap* avant de l'utiliser :

```
/sbin/mkswap /dev/hdb3
action "Activating swap partition" swapon –a
```

Naturellement, on adaptera cette ligne en fonction du contenu de `/etc/fstab`.

Lorsque le système démarre, il monte d'abord la partition racine en lecture seule, puis la remonte ensuite en lecture-écriture. Nous supprimons cette ligne en la mettant en commentaire :

```
# action "Remounting root filesystem in
read-write mode" mount –n –o remount,rw /
```

Lorsque `/sbin/mount` est invoqué pour monter une partition, cet utilitaire écrit dans le fichier `/etc/mtab` le résultat de son travail, ceci pour que l'on connaisse facilement l'état des systèmes de fichier. Comme `/etc` est en lecture seule, il faut supprimer cette tentative d'écriture.

Cela s'effectue en ajoutant l'option –n à `mount` dans les lignes suivantes :

```
mount –n –f –t proc /proc /proc
mount –n –f /
action "Mounting local filesystems" mount –n –a –t nonfs,smbfs,noncpfs,proc
```

On pourra toujours consulter le fichier `/proc/filesystems` pour obtenir le même résultat que la lecture de `/etc/mtab`.

Enfin, on supprime les tentatives d'écriture dans des fichiers se trouvant sur la partition en lecture seule, à commencer par `/etc/mtab` :

```
# >/etc/mtab
# echo ${HOSTNAME} > /etc/HOSTNAME
```

et

```
# cat > /boot/kernel.h << EOF
#    #ifndef __BOOT_KERNEL_H

#   #endif
#  EOF
```

A présent, nous devons introduire, dans ce script `/etc/rc.sysinit`, le formatage de `/var` et la mise en place de ses sous-répertoires. Nous ajoutons donc les lignes suivantes :

```
/sbin/mke2fs /dev/hdb2
mount /var
mkdir /var/authdir
mkdir /var/authdir/authfiles
mkdir /var/lib
mkdir /var/lib/xkb
mkdir /var/lock
mkdir /var/lock/console
mkdir /var/lock/subsys
mkdir /var/log
mkdir /var/run
mkdir /var/spool
mkdir /var/spool/at
mkdir /var/spool/cron
mkdir /var/tmp
chmod 777 /var/tmp
chmod +t /var/tmp
umount /var
```

Le plus simple est d'ajouter ces lignes avant le montage automatique de toutes les partitions.

Répertoires et liens

Les sous-répertoires de `/var` sont créés à chaque redémarrage par les lignes que nous avons insérées dans `/etc/rc.d/rc.sysinit`. On remarquera que nous y avons placé un répertoire `/var/tmp` avec les droits de lecture, écriture et parcours pour tous, ainsi que le *sticky-bit* qui ne permet la modification d'un fichier que par son propriétaire. Nous créons un lien symbolique depuis `/tmp` vers ce répertoire :

```
ln –sf /var/tmp /tmp
```

Quelques autres liens sont également nécessaires, essentiellement parce qu'en raison de choix de conception douteux, certaines applications placent des fichiers temporaires dans de mauvais répertoires.

Si vous utilisez les extensions clavier XKB de X-Window :

```
cd /usr/X11R6/lib/X11/xkb
ln –sf /var/lib/xkb compiled
```

Si vous utilisez `xdm`, celui-ci a besoin d'un endroit pour stocker des fichiers d'authentification :

```
cd /etc/X11/xdm/
ln –sf /var/authdir authdir
```

De son côté, `kdm` utilise `/etc/X11/xdm/authdir/authfiles`, ce qui explique que nous ayons créé ce sous-répertoire sur `/var/authdir`.

Enfin, il faut traiter le cas particulier du démon `syslogd`, qui sert à enregistrer ou diffuser des messages en provenance de diverses applications. Ce mécanisme est particulièrement important ici, car les applications ne pouvant pas écrire leurs messages d'erreur sur la sortie standard (les démons par exemple), l'emploient pour nous indiquer leurs difficultés. La plupart des informations sont mémorisées dans le fichier `/var/log/messages`. Pour communiquer avec les applications désirant enregistrer des messages, le démon `syslogd` crée une socket `/dev/log`, sur laquelle il se met en écoute. Le répertoire `/dev` étant en lecture seule, ceci nous pose un problème. Heureusement, `syslogd` accepte, lors de son initialisation, une option –p permettant de préciser l'emplacement où la socket doit être créée. On modifiera donc le fichier `/etc/rc.d/init.d/syslog`, plus précisément la ligne suivante :

```
daemon syslogd –m 0 –p /var/log/syslogd_sock
```

Toutefois cela ne serait pas suffisant, car les autres applications ne sauraient pas où trouver la socket de contact de `syslogd`. Nous devons donc ajouter un lien symbolique permettant d'utiliser `/dev/log` :

```
ln –sf /var/log/syslogd_sock /dev/log
```

Premier essai

Avant de relancer la machine, une dernière modification s'impose, concernant le fichier `/etc/rc.d/rc.local`. Dans ce script en effet, on écrit quelques informations dans le fichier présentant la bannière de connexion. Nous devons donc mettre ces lignes en commentaire :

```
# echo "" > /etc/issue
# echo "$R" >> /etc/issue

# echo >> /etc/issue
```

Je conseille tout d'abord de tenter une initialisation en mode multi-utilisateur non-graphique (`init 3` sur Red-Hat) afin de détecter déjà tous les problèmes hors X-Window. On prendra soin de désactiver tous les services non utilisés (`sendmail`, `nfs`, `lpd`) dans un premier temps.

Il faut alors noter tous les messages d'erreurs survenant lors de l'initialisation, et en rechercher la cause, pour la corriger. On pourra généralement régler le problème à l'aide d'un lien symbolique vers un sous-répertoire de `/var`, comme nous l'avons fait avec `syslogd`.

Avant de modifier un fichier système, il ne faut pas oublier d'invoquer

```
mount / -o rw,remount
```

sous peine de ne pouvoir sauvegarder les corrections.

Une fois la configuration au point en mode non-graphique, on pourra tenter de démarrer X11, puis d'initialiser directement la machine sur une connexion `xdm`, comme nous l'avons fait dans le précédent article.

Le système de gestion des paquetages Red-Hat emploie une base de données `/var/log/rpm`. Pour éviter de perdre ce fichier à chaque redémarrage, lors du formatage de `/var`, je conseille de le déplacer dans `/usr/local/lib/rpm`, et de créer un lien symbolique :

```
ln –sf /usr/local/lib/rpm /var/log/rpm
```

Bien sûr ce lien doit être recréé à chaque redémarrage. Naturellement, avant d'ajouter ou de supprimer un paquetage, il faut remonter la partition racine en lecture-écriture, ce qui permet de modifier cette base de données.

Extensions possibles

La configuration décrite ici est très limitée. En fait c'est celle que j'emploie pour transformer une machine linux en station X-Window servant uniquement à se connecter sur d'autres ordinateurs pour travailler. On n'utilise pas de données persistantes sur cette machine (`/home` par exemple).

La première chose qu'il peut être nécessaire d'ajouter, c'est un répertoire accessible en lecture-écriture, afin d'y stocker des données ayant une longue durée de vie, comme celles appartenant aux utilisateurs. J'ai rencontré cette situation dans une installation où chaque station utilisateur devait pouvoir résister à de fortes contraintes de manipulation hasardeuse, tout en disposant de fichiers personnels pour les utilisateurs (fichiers de configuration de l'application principale utilisée sur le système).

La seule solution viable, est d'utiliser un serveur NFS central, sur lequel résident les données personnelles, et de monter par le réseau ces répertoires sur les stations utilisateurs. Le serveur NFS devra naturellement être protégé par un onduleur, et être sous la responsabilité d'un administrateur qui appliquera correctement les procédures d'arrêt.

L'avantage de ce système est que l'extinction inopinée d'une station ne joue pas sur l'intégrité des données, puisqu'elles sont gérées par le système central.

Bien sûr, on ne peut pas employer de serveur NFS dans le cas d'une machine embarquée dans un véhicule par exemple, mais là pas de miracle possible, si des partitions doivent être utilisées en écriture et si l'alimentation électrique risque d'être interrompue brutalement, il faut obligatoirement se tourner vers un système d'accumulateurs ou de batteries supplémentaires.

Notre description a été volontairement très simplifiée, et il faut prendre en compte tous les démons (`sendmail`, `httpd`, `lpd`) qui nécessitent des espaces de stockage temporaires (souvent dans `/var/log/`). On restreindra au maximum le nombre de ces applications système, en n'installant que celles qui sont indispensables pour le rôle attribué à la machine.

Conclusion

Plutôt qu'une recette pour configurer sa machine, j'espère avoir ici fourni un sujet de réflexion et de curiosité pouvant susciter l'envie d'explorer les possibilités d'une telle manipulation. Encore une fois, nous pouvons admirer la souplesse du système Linux, qui nous permet ainsi de créer des stations à la configuration durcie, que l'on peut laisser sans crainte dans des environnements sans surveillance.

Christophe Blaess < ccb@club-internet.fr>

http://perso.club-internet.fr/ccb/

Auteur de l'ouvrage ‘‘ *Programmation système en C sous Linux* ’’ aux éditions Eyrolles, et coordinateur des traductions françaises des pages de manuel pour Linux.