Systèmes d'exploitation

Module UE142 : administration de systèmes

Licence professionnelle

Pierre Nerzic

IUT de Lannion

Partie A: Constitution et fonctionnement du système Linux

1 - Qu'est-ce que Linux ?

C'est un ensemble complexe de logiciels permettant de :

- faire fonctionner un PC (en tant que machine de calcul et non de chauffage personnel), de l'utiliser pour travailler...

- programmer des logiciels : librairies (dll) de fonctions à disposition.

Il est composé de plusieurs parties :

- un chargeur de système : MBR + lilo, grub pour lire les fichiers du système et les mettre en mémoire,

- un programme principal appelé noyau, accompagné d'un système de fichier contenant les compléments du noyau : modules (drivers) et librairies,

- des programmes complémentaires pour gérer les services : shells, commandes, services réseau et autres.

2 - Démarrage du système

Un système d'exploitation est un logiciel extrêmement complexe, composé de nombreux fichiers : noyau et ses modules, librairies, services. Il ne peut plus être placé en ROM comme dans les premiers ordinateurs. Son chargement en mémoire est une opération complexe.

De même, son arrêt ne peut plus se faire en éteignant la machine directement. Il faut prendre des précautions pour ne pas risquer de détruire les données.

Ce chapitre décrit les différentes phases du démarrage et de l'arrêt du système.

a) Démarrage initial d'un PC

Phases du démarrage (BOOT) :

i) Le BIOS

Quand on allume le PC, le processeur démarre un programme situé sur une EEPROM de la carte mère appelé BIOS (Basic Input Output System).

Le BIOS fait quelques tests (nommés POST : pré operating system tests) sur le matériel (quantité de mémoire, disques durs installés, périphériques PCI) – à ce stade, on peut arrêter le BIOS pour le configurer (selon les bios, ex : touche DEL).

L'un des périphériques est désigné « disque de démarrage » soit par la configuration du BIOS, soit par l'utilisateur (touche F11 ou entrée, selon les bios).

Le BIOS charge en mémoire le premier secteur du disque désigné pour le démarrage. Ce premier secteur est appelé MBR (Master Boot Record) sur un disque dur et Secteur de démarrage ou d'amorçage (Boot sector) sur une disquette.

Le MBR contient d'une part le programme initial et d'autre part la table des partitions. Ce programme initial commence par vérifier la table des

partitions puis charge le premier secteur de la partition notée active (bootable), voir Master Boot Record.htm.

Dans tous les cas, on se retrouve avec le contenu du secteur de boot dans la mémoire, c'est un tout petit programme, environ 200 octets. Le BIOS le démarre.

ii) Démarrage : lilo, grub ou autre

Ce tout premier programme est très « figé », son rôle et ses possibilités sont des plus limités : il ne connaît pas les formats de disques (File systems), il ne sait donc pas retrouver des fichiers dans une arborescence. La seule chose qu'il peut faire, c'est lire des blocs dont il connaît les numéros. Il doit donc laisser la place à un programme plus élaboré pour charger les nombreux fichiers du système.

Donc le secteur de boot charge un second chargeur de système, par exemple lilo, grub sur linux, syslinux sur knoppix.

Lilo et grub donnent le choix à l'utilisateur du système à charger. Ces logiciels n'ont en général aucune connaissance des formats de disque (ext2, reiser, fat, ntfs), ils ne savent lire que des suites de secteurs en

utilisant les routines du BIOS (INT 13) mais ils savent où sont exactement situés les fichiers du système.

Quand on configure lilo, ce dernier met à jour le MBR en lui indiquant où se trouvent les blocs du noyau linux : un fichier spécial, formé de blocs consécutifs contient la liste des blocs à charger en mémoire.

Pour résumer :

BIOS --> MBR --> lilo puis lilo charge le noyau.

b) Lancement de Linux

Le noyau Linux est d'abord chargé sous forme d'une « image mémoire ». Il s'agit d'une copie monolithique du contenu exact de la mémoire centrale : le noyau du système = les programmes nécessaires pour faire fonctionner la machine : gestion de processus, mémoire et fichiers. Cette image ressemble à un fichier core qui serait parfaitement fonctionnel. Il s'appelle en général vmlinux sur linux (vmlinuz est sa version compressée autoextractible).

On ne peut pas charger le reste du système (drivers, dll, fichiers de config) directement, c'est trop compliqué, il y a trop de fichiers

différents. Pour charger des fichiers, il faut mettre en place un système de fichier (File system) ; cela suppose un formatage spécifique du disque ou bien la possibilité de lire différents formats ; cela suppose aussi la présence d'un programme spécifique pour charger les différents fichiers du système (savoir lesquels charger et savoir comment les charger : accès disques, édition des liens).

Cas concret : la carte mère contient des disques SATA RAID. Comment faire pour que le noyau puisse accéder aux fichiers de ces disques si le driver SATA se trouve quelque part sur ces disques ? Il faut intégrer le driver au noyau ou fournir le driver au moment du boot, ça peut faire beaucoup de drivers, dont seulement certains vont servir.

Pour simplifier tout ça, en général (liveCD p. ex. ubuntu, knoppix,), on charge aussi un mini système de fichiers Unix qui est placé en mémoire : on le qualifie de « RAM Disk » : les fichiers sont dans des zones mémoire et non pas dans des blocs sur un disque réel. Ce système de fichiers contient une petite arborescence unix contenant les fichiers indispensables au noyau. Ce disque mémoire est nommé initrd.img.

Donc, pour résumer, le compilateur du système d'exploitation a préparé un fichier linéaire contenant une image du système vmlinuz + initrd.img. Cette image, une fois chargée en mémoire par Lilo, est lancée et devient le noyau du système d'exploitation.

Pour résumer :

BIOS --> MBR --> lilo --> noyau puis le noyau initialise le système.

i) Initialisation du noyau

La première tâche du noyau est d'initialiser les périphériques et de vérifier qu'ils fonctionnent. Cette tâche est effectuée par le script /linuxrc sur le disque mémoire initrd (ce fichier disparaît ensuite car le disque RAM est démonté une fois le système initialisé).

Ensuite le noyau monte le volume racine spécifié par la directive lilo **`root=`**. En lecture seule : message affiché « **`VFS: Mounted root (ext2 filesystem) readonly.`** »

Pour finir, avant de se retirer de la scène et de servir uniquement de support au librairies, le noyau lance le processus **`/sbin/init`** (de PID 1).

ii) Le programme /sbin/init et sa configuration /etc/inittab

Ce programme continue l'initialisation du système. On passe maintenant à un haut niveau qui permet de tout configurer très facilement : c'est fait par des scripts shell. Le but de ces scripts est de lancer les démons du système : gestion de tous les services comptes, imprimante, réseau…

NB : ce cours présente tout d'abord une version du système qui devient obsolete avec le temps : la version System V basée sur le fichier /etc/inittab. On rencontre maintenant upstart dont il est question plus loin.

Initialisation globale

Init consulte le fichier /etc/inittab dans lequel on trouve des directives « lancer tel service et attendre », « lancer tel autre service indéfiniment »… La syntaxe est décrite plus loin.

Par exemple, au boot /etc/inittab spécifie le script à exécuter : /etc/rc.d/rc.sysinit sur redhat ou /etc/init.d/rcS sur debian. Le premier est un script bash qui :

- initialise quelques variables : PATH...,
- monte le système de fichiers /proc,
- démonte le système /initrd,
- configure quelques paramètres du noyau,
- configure des périphériques (USB, PNP, port série, SCSI...),
- vérifie l'état des volumes à monter (la présence du fichier /forcefsck oblige à vérifier les volumes),
- remonte la racine en lecture-écriture,
- charge les modules nécessaires au noyau (son, raid, réseau),
- nettoie certains fichiers de /var,
- met en place le mode DMA sur les disques et CD-Rom.

Ensuite init lance les services : réseau, serveurs, comptes…

Initialisation des services

Les administrateurs ont souhaité distinguer plusieurs modes de fonctionnement du système : le mode normal multiutilisateurs, un mode monoutilisateur pour l'administration, un mode sans réseau, un mode d'extinction du système, un mode redémarrage. Ces modes de fonctionnement sont appelés (très à tort) niveaux d'exécution (runlevel).

Les « niveaux » principaux sont numérotés de 0 à 6 et il en existe quelques autres comme les niveaux S ou s qui correspondent à des états plus spécifiques. Les niveaux les plus fréquemment employés (c'est l'administrateur qui décide de leur nom et de leur contenu) sont :

0 : arrêt du système
S : monoutilisateur (pas de service de connexion ailleurs que sur la console)
1 : multiutilisateur (service de connexion), pas de réseau
2 : multiutilisateur (service de connexion), réseau

3 : multiutilisateur (service de connexion), réseau et xdm (connexion X11)
6 : redémarrage du système

NB : il n'y a pas de relation d'ordre entre les niveaux, il s'agit seulement d'un identifiant.

C'est le programme /sbin/init qui choisit ce qu'il faut faire en fonction du runlevel en cours, par exemple lancer ou non les services réseau, les connexions distantes...

Syntaxe du fichier `/etc/inittab` (`man inittab`)

Le fichier **`/etc/inittab`** définit d'une part le niveau au démarrage : ligne **`id:2:initdefault:`** ici on démarre avec le niveau 2. Le changement de niveau est provoqué par la fin des processus lancés à ce stade, un appel à la commande **`/sbin/telinit`**, ou d'autres commandes (ex : shutdown).

D'autre part, ce fichier indique ce qu'on doit faire pour chaque niveau d'exécution. Il est composé de lignes :

`label:runlevel(s):circonstance:processus`

- label = un symbole unique pour identifier cette ligne
- runlevel(s) = liste des niveaux concernés (ex : 123 = les niveaux 1, 2 et 3)
- processus = la commande à lancer dans cette circonstance et dans ce(s) runlevel(s).
- circonstance ou action à faire sur le processus. Il y en a de très nombreuses, voici une sélection :
  - **`initdefault`** = indique le niveau d'exécution de démarrage du système. Par exemple : **`id:3:initdefault:`** indique de débuter au niveau 3.
  - **`respawn`** = relancer en cas de fin,
  - **`wait`** = lancer le processus et attendre sa fin,
  - **`once`** = lancer une seule fois,
  - **`boot`** = lancer lors du boot,
  - **`sysinit`** = lancer uniquement lors du boot et attendre la fin,

Voici un exemple (redhat) de fichier **/etc/inittab** :

```
id:3:initdefault:                                  niveau à adopter au tout début
si::sysinit:/etc/rc.d/rc.sysinit                   script à lancer au tout début
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3                           lancer les services et attendre
l3:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6
2:235:respawn:/sbin/mingetty tty2
3:235:respawn:/sbin/mingetty tty3                  relancer indéfiniment ce programme
x:5:respawn:/etc/X11/prefdm -nodaemon
```

iii) Upstart = remplacement de /etc/inittab

L'ensemble upstart remplace l'ensemble init classique. Ce lien http://upstart.ubuntu.com/ présente le dispositif. C'est basé sur le paradigme événement – action : on définit des relations : si tel événement survient, faire telle action.

Donc, au lieu de définir une sorte de script comme inittab, on définit toute une liste de relations : dans tel runlevel démarrer ceci...

Le répertoire /etc/event.d contient toutes les définitions pour upstart. Tous ces fichiers qu'on appelle des jobs sont examinés au boot et selon le cas de figure : tel ou tel runlevel ou événement, on fait ceci ou cela.

La syntaxe est relativement claire :

- soit on lance une commande : exec commande
- soit on exécute un script shell inclus dans le job : script... end script

Ensuite, on indique quand les scripts doivent démarrer ou s'arrêter. Si ces informations ne sont pas données ou mises en commentaire, le script ou la commande n'est pas exécuté.

- start on *condition*
- stop on *condition*

Les conditions sont :

- des runlevels : le mot clé runlevel suivi des noms des runlevels (numéro ou S),

- des états particuliers : p. ex. startup, stalled
- des actions spécifiques (control-alt-delete),
- des changements sur d'autres jobs : stopped *autrejob*, started *autrejob*,
- ...à venir dans le futur...

Bien noter que upstart est en développement et que la syntaxe n'est pas figée définitivement.

L'état des jobs est vu par la commande : sudo initctl list

Le runlevel par défaut est déterminé par le job rc-default dont l'action est d'exécuter la commande telinit, cette dernière change le runlevel.

Le futur de upstart devrait permettre de remplacer les démons inetd, cron et anacron (l'un des prochains cours).

iv) Le programme /etc/rc.d/rc et les runlevel

Le programme **`/etc/rc.d/rc`** est le lanceur des services et il est appelé par init via inittab ou upstart. Avant de regarder ce que fait exactement **`/etc/rc.d/rc`**, étudions les services concernés.

Les services et les démons

De nombreux aspects du système ne sont pas intégrés au noyau. Ils sont gérés par des processus séparés. C'est le cas de l'imprimante (lpd), de la connexion des utilisateurs (getty), de tous les services réseau, du gestionnaire des tâches répétitives (at, cron)...

Un service Unix est constitué de différentes choses :

- d'un ou plusieurs processus qui tournent en boucle infinie en attente d'ordres (les démons), par exemple /usr/sbin/lpd pour l'imprimante.
- des fichiers de configuration qui sont lus au lancement du démon, par exemple /etc/lpd.conf pour l'imprimante.
- des fichiers d'état (contenant par exemple le PID du démon), /var/run/atd.pid pour le démon at.

Le lancement complet d'un service est assez compliqué, il ne suffit pas de lancer le processus en arrière-plan, il faut souvent tester l'existence de fichiers, la configuration de la machine, la présence d'autres démons...

De même, l'arrêt du service est également compliqué, il ne suffit pas de tuer le processus démon, il faut effacer les fichiers temporaires, préparer un redémarrage futur, etc.

Pour ces raisons, toutes ces phases sont programmées dans un script recevant un seul paramètre : « start » ou « stop » qui indique au script de démarrer le service, de l'arrêter. Il existe d'autre mots clés p. ex. pour consulter l'état du service.

Tous ces scripts sont dans **`/etc/init.d`**. NB: c'est un lien logique vers **`/etc/rc.d/init.d`**. Par exemple, **`/etc/init.d/lpd`** permet de commander le fonctionnement du démon gérant l'imprimante.

C'est l'un des éléments importants sur lesquels l'administrateur intervient.

Voici la liste des mots-clés de commande des services :

- **`start`** : démarre le service
- **`stop`** : arrête le service (tue les processus, supprime les fichiers...)

Ces deux mots-clés sont connus de tous les services et sont gérés par le gestionnaire de runlevel **`rc`** (voir ci-dessous). Les mots-clés suivants complètent les possibilités pour l'administrateur, mais certains services ne les connaissent pas, le mieux est de consulter leur source bash :

- **`status`** : affiche l'état du service
- **`restart`** : arrête puis redémarre le service, par exemple en cas de plantage
- **`reload`** : recharge le service, par exemple si on a changé l'un de ses fichiers de configuration, ça ne tue pas le démon mais on lui envoie un signal particulier qu'il sait reconnaître comme tel.

ex:

- **`/etc/init.d/lpd restart`**
- **`/etc/init.d/smb reload`**
- **`/etc/init.d/mysqld stop`**

Les services et `/sbin/init`

Le démarrage (**`/etc/init.d/`*`service`*` start`**) et l'arrêt (**`/etc/init.d/`*`service`*` stop`**) ne sont pas faits directement par **`init`**. Ce serait trop lourd de modifier **`/etc/inittab`**. C'est un autre logiciel : **`/etc/rc.d/rc`** qui se charge de cela. L'architecture de cet ensemble permet de changer rapidement un paramètre (ajouter ou retirer un service s à chaud) mais ça reste très lourd et totalement perfectible (d'ailleurs, ça n'arrête pas de changer légèrement d'une version/distribution d'Unix à l'autre – ce cours présente la variante Redhat issue de System V.3 similaire aux Debian/Ubuntu). C'est perfectible parce que par exemple il faut indiquer à chaque runlevel les services qu'on arrête et ceux qu'on démarre.

Les répertoires **`/etc/rc*.d`** contient différents éléments pour gérer les services en fonction des runlevels :

- le script **`/etc/init.d/rc`** déjà présenté qui gère le démarrage et l'arrêt des services en fonction des changements de runlevel.

- autant de répertoires **`/etc/rc`***`N`***`.d/`** que de runlevels connus où *N* désigne le runlevel concerné. Le contenu de ces répertoires définit ce

qu'on fait quand on arrive dans ce runlevel. Par exemple, **/etc/rc5.d** définit ce qu'on fait quand on passe dans le runlevel 5.

NB : selon les systèmes, c'est dans **/etc/rc*N*.d** ou **/etc/rc.d/rc*N*.d**.

Dans chacun de ces répertoires, on trouve des liens logiques vers les services. Par exemple, on trouve des liens vers lpd, atd, pppd, mysqld...

Ces liens logiques ont un nom particulier. Ils ne s'appellent pas **lpd** directement, mais **S60lpd** par exemple. Cette astuce permet de définir quels services on arrête et lesquels on lance. Ce n'est pas une excellente solution : on aurait pu imaginer deux sous-répertoires ou bien deux listes texte ou encore mieux : un véritable suivi des services, mais comme c'est programmé en bash, c'est plus simple de ruser avec des noms de fichiers ; ça permet aussi de définir dans quel ordre on les arrête et active.

La syntaxe exacte des noms de liens est **[KS]*NNnom***. Par exemple, **S95atd**, **K35smb**. Il y a deux familles de noms. La famille K est celle des services qu'il faut arrêter quand on arrive dans ce runlevel (c'est pour les services actifs dans le précédent runlevel et qu'on ne veut plus ici). La famille S est celle de ceux qu'il faut activer. En principe, quand on

entre dans un runlevel, on arrête tous les services du précédent runlevel, mais évidemment, on n'arrête pas un service qui est présent également dans le nouveau runlevel.

Le numéro NN du lien permet de donner un ordre de précédence dans les services. K12mysqld sera arrêté avant K35smb, et S95atd sera démarré après S80sendmail.

C'est le script **`/etc/init.d/rc`** qui gère ces arrêts et ces démarrages selon le runlevel. C'est tout simple : il envoie stop aux services K* et start aux services S*.

Et c'est init (par la configuration de inittab, ex: l5:5:wait:/etc/init.d/rc 5) qui appelle rc avec le bon numéro de runlevel en paramètre.

Exemple de configuration

Voici un petit exemple d'un service inutile. Admettons qu'on veuille transformer en service le programme bash suivant :

```
#!/bin/bash
trap "echo -n signal 2 reçu a >> $1 ; date >> $1" 2
trap "echo -n mort a >> $1 ; date >> $1 ; kill -9 $$" 3
```

```
while true ; do wait ; done
```

Ce programme tourne en boucle infinie en attente de signal. Quand il reçoit le signal 2, il écrit ‘signal 2 reçu à …’ dans le fichier log fourni en premier paramètre et quand il reçoit le signal 3, il écrit ‘mort à …’ et se termine. L’instruction trap permet de faire une action quand on envoie un signal au processus par kill. Attention, pour le TP, on n’utilisera pas les signaux 2 et 3 car ils sont interceptés.

Exemple d’utilisation :

```
$ mondemon_d ./log &
[1] 10381
$ kill -2 10381
$ kill -3 10381
```

On voudrait que ce démon soit lancé dans le runlevel 3. Voici ce qu’il faut faire :

- Programmer son gestionnaire de service : **/etc/init.d/mondemon**

```
#!/bin/sh
# script /etc/init.d/mondemon
```

```
check_status()
{
    if [ $? = 0 ]
    then echo "[OK]"
    else echo "[FAILED]"
    fi
}

case "$1" in
    start)
       echo -n "Demarrage de mon demon..."
       if [ -f /var/run/mondemon.pid ]
       then echo '[RUNNING]'
       else
          mondemon_d /tmp/log &
          check_status
          echo $! > /var/run/mondemon.pid
       fi
       ;;
    stop)
```

```
    echo -n "Arret de mon demon..."
    if [ ! -f /var/run/mondemon.pid ]
    then echo '[ABSENT]'
    else
       # envoyer le signal 3 au processus
       /bin/kill -3 `cat /var/run/mondemon.pid`
       check_status
       rm -f /var/run/mondemon.pid
    fi
    ;;
*) echo "Usage: $0 {start|stop}"
    ;;
esac
```

Ne pas oublier de le rendre exécutable. Le tester avant de l'installer pour de bon.

Note : il est conseillé de partir d'une copie de /etc/init.d/skeleton.

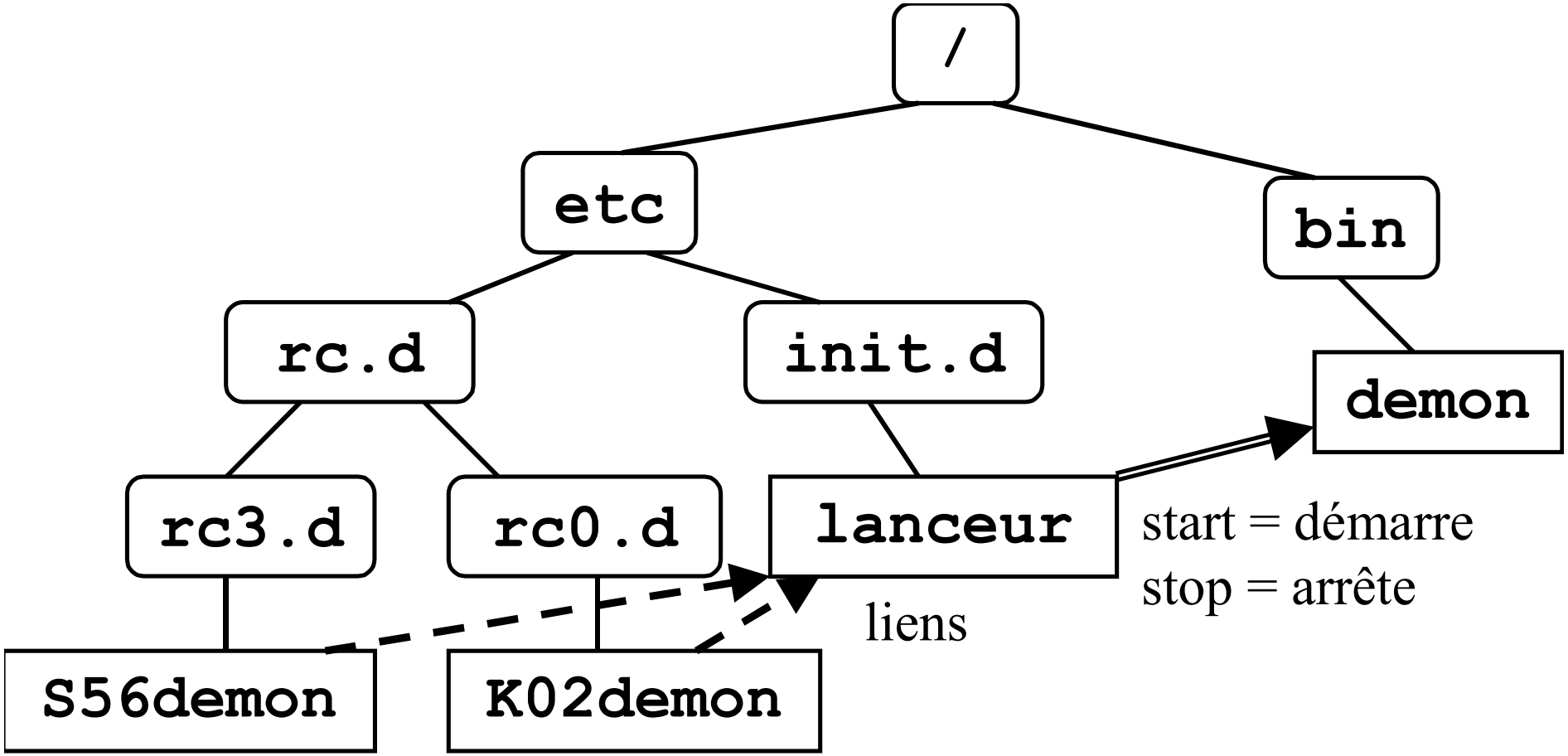
- Créer des liens dans les répertoires des runlevels concernés. On veut démarrer ce démon dans le runlevel 3 et l'arrêter dans les runlevels 0 et 6.
  - `cd /etc/rc3.d ; ln -s ../init.d/mondemon S99mondemon`
  - `cd /etc/rc0.d ; ln -s ../init.d/mondemon K90mondemon`
  - `cd /etc/rc6.d ; ln -s ../init.d/mondemon K90mondemon`

Sur Debian/Ubuntu, la commande update-rc.d permet d'ajouter ou supprimer des services de la liste d'un ou plusieurs runlevels.

```
update-rc.d -f name remove
update-rc.d name defaults [NN | NN-start NN-stop]
update-rc.d name start|stop NN runlevel runlevel ...
```

name est le service qu'on veut gérer. C'est donc un script de /etc/init.d. NN est l'ordre de précédence de lancement ou arrêt du script.

v) Résumé



Le script /etc/init.d/rc N lancé par /sbin/init dans le runlevel *N* parcourt le répertoire /etc/rc*N*.d, on y trouve des fichiers appelés S* ou K*. Ce sont tous des liens vers des scripts situés dans /etc/init.d. Les S* sont exécutés avec le paramètre start, les K* avec le paramètre stop. Ca lance ou arrête les services concernés.

vi) Autres scripts utiles et options du boot

Selon les systèmes, on peut aussi rencontrer d'autres scripts lancés au démarrage : rc.sysinit, rc.local. Soit ils sont lancés par inittab (ou upstart), soit c'est un lien dans un runlevel.

Un point très intéressant est qu'ils reçoivent les options de démarrage sous forme de variables d'environnement. En effet, la syntaxe d'une option est « var=valeur », par exemple . Ils peuvent aussi consulter /proc/cmdline et faire getopt mais c'est plus compliqué.

3 - Arrêt du système Linux

a) Arrêt normal

- **`shutdown -hrf [heure ou +nbminutes]`**

La commande shutdown permet d'arrêter le système proprement à l'heure indiquée ("now" par défaut) : elle fait passer dans le runlevel 0 ou 6 selon l'option -h (extinction du PC) ou -r (reboot).

Cette commande fait en sorte que de nouvelles connexions soient impossibles, elle prévient les utilisateurs que le système va être arrêté leur permettant de sauver leur travail.

L'option -f permet de signaler au prochain redémarrage que les disques sont en bon état car leur démontage s'est bien passé.

b) Arrêt involontaire – plantage grave

En cas de coupure de courant ou autre, le système est dit "mal arrêté". Les processus ont été tués sans avoir eu le temps de sauver les données en mémoire sur disque.

Pour ce qui est des démons, les scripts /etc/init.d/ savent en général réinitialiser leur travail (effacer les fichiers temporaires...).

Mais parfois, les systèmes de fichiers en Read/Write sont abimés car le démon sync n'a pas pu vider les buffers et le démontage n'a pas eu lieu. Par conséquent, la commande de réparation fsck sera forcée au prochain démarrage.

Pour que le système sache que l’extinction n’a pas été propre, l'astuce est de créer dès le démarrage un fichier /.autofsck qui, s'il est présent

c'est qu'on ne l'a pas effacé (LaPalisse), si on ne l’a pas effacé c’est qu’on a mal éteint car normalement ce fichier est enlevé par la commande /etc/init.d/halt lancée par le runlevel 0.

Partie B: Réglage et maintenance du noyau Unix

1 - Noyau Linux

Le noyau unix/linux est écrit quasiment entièrement en C, ce qui permet :

- de le porter sur différentes architectures
- de le modifier comme on le souhaite sans trop de difficultés.

En général, on ne récrit pas le système, mais on ajuste seulement ce qu'il contient. En effet, le noyau linux est composé de

- Une partie noyau pure : par exemple, le gestionnaire de processus (scheduler), le gestionnaire de mémoire virtuelle...
- Une partie modules : les programmes spécifiques pour faire fonctionner les périphériques (drivers), des modules pour gérer des

possibilités optionnelles (autres file systems, fonctions de calcul optionnelles : cryptage, nombres aléatoires…)

La partie modulaire est configurable : on peut décider que le noyau possède ou non tel ou tel élément. Plus précisément, pour un élément donné, on peut décider qu'il

- fait partie permanente du noyau « Y »
- qu'il ne sera chargé en mémoire que si le besoin apparaît, « M »
- qu'il est absent définitivement. « N »

La différence entre les trois cas : un module Y est compilé dans le noyau, son code machine fait partie de vmlinuz, ce sont des procédures et des fonctions normales ; un module M est compilé mais sous forme d'un fichier objet.o qui est chargé à la demande, ce fichier est placé dans initrd.img ou dans /lib/modules/… (voir + loin) et les fonctions sont appelées par liaison dynamique ; un module N n'est pas compilé du tout.

Les modules Y sont plus rapidement chargés que les modules M (car déjà là), c'est pourquoi il vaut mieux mettre tout ce dont on a besoin en

permanence en type Y, par contre le noyau grossit d'autant. Il est inutile de mettre trop de modules M si on n'a pas le matériel correspondant (ex : radio amateur...). A l'IUT, tout est en Y ou en N, il n'y a aucun module optionnel.

Les commandes suivantes permettent de savoir de quoi est composé le noyau à un instant donné.

- /sbin/lsmod : permet de lister l'ensemble des modules présents en mémoire
- /sbin/modprobe : permet d'insérer un module en mémoire, ainsi que toutes ses dépendances. Utilise l'autre commande /sbin/insmod.
- /sbin/rmmod : permet de supprimer un module de la mémoire.
- /sbin/depmod : reconstruit le fichier modules.dep qui indique les dépendances entre les modules, par exemple un driver de webcam a besoin de l'USB et de la vidéo. Cette liste de dépendances est utilisée par modprobe pour que tout fonctionne bien.

2 - Configurer et compiler le noyau

Configurer le noyau consiste à choisir les modules Y et M. Il y en a un nombre très important et il y a des dépendances entre eux. Par exemple, le module d'accélération graphique drm fait appel au module agpgart. La lecture de la documentation est indispensable. Parfois même, il faut aller voir le source C des modules pour savoir s'ils vont être utiles.

Voici la procédure à suivre pour créer un noyau personnel :

- Obtenir les sources du noyau, en général dans /usr/src/*version*. Il faut parfois employer des patches. L'expérience montre que c'est très difficile d'arriver à patcher. Le mieux est d'avoir les fichiers entiers de la version souhaitée.
- Configurer le noyau : choisir les modules Y, M et N. Il existe une commande make xconfig pour cela. Cette commande crée/édite le fichier .config qui décrit tous les modules du futur noyau.
- Compiler le noyau : il suffit de taper **`make bzImage modules install module_install`** pour que toutes les opérations soient faites.

- Installer le noyau pour le démarrage : configurer Lilo ou grub (§ suivant)

Le cours continuera en TP.

3 - Configurer le chargement du noyau

a) Boot normaux

Il y a principalement deux chargeurs de linux sur disque dur : lilo et grub.

On rappelle que le chargement consiste à lire le noyau du disque et à le mettre en mémoire. Voici les éléments nécessaires :

- les fichiers du noyau : vmlinuz et initrd sont situés dans dans /boot ou /tftpboot
- un fichier de configuration, ex /etc/lilo.conf donne la liste des systèmes et variantes à lancer ; cette liste est affichée sous forme de menu au démarrage.

- un logiciel installé dans le MBR du disque ou le secteur de boot de la partition active consulte le fichier de configuration et charge les fichiers du noyau au choix de l'utilisateur.

Il suffit d'éditer le fichier de configuration, rajouter les items voulus. Voir la séance de TP pour les options de configuration.

b) Boot spéciaux

Il existe un chargeur, pxelinux ou grub qui sont capables de se connecter à des serveurs DHCP puis TFTP pour télécharger un noyau au lieu de le lire auprès du disque. Ce dispositif fera l'objet d'un grand TP dans les semaines à venir.