

Objectifs

Ce TP va, dans sa première partie, permettre de mettre en évidence un certain nombre de choses, concernant les possibilités d'un vrai système d'exploitation multi tâches et multi utilisateurs.

Dans la seconde partie, nous aborderons un peu plus en profondeur les notions de « services » (« Daemons » dans la terminologie UNIX), de « serveurs », ainsi que de « sockets ».

Etape préliminaire

Installez votre disque dur sur votre machine, démarrez Linux, ouvrez une session « root » et vérifiez que vous avez un serveur ssh qui fonctionne.

Pour ce faire, utilisez la commande :

```
ps aux | grep sshd
```

Vous devez obtenir dans la réponse une ligne ayant cette allure :

```
root    3086  0.0  0.6  5104 1564 ?  Ss   10:32   0:00 /usr/sbin/sshd
```

Seul ce qui est en gras a pour l'instant de l'importance. Nous verrons plus loin la signification de cette ligne un peu plus en détails.

Vérifiez également que vncserver est installé et en état de fonctionner :

```
vncserver
```

Cette commande doit vous retourner une réponse dans ce genre :

```
New 'X' desktop is <nom de votre machine>:1
```

```
Starting applications specified in /etc/X11/Xsession  
Log file is /root/.vnc/<nom de votre machine>:1.log
```

Enfin, vérifiez que votre configuration IP est correcte, c'est-à-dire statique, avec :

- une adresse IP de la forme 172.16.51.<le numéro de votre machine>>,
- un masque de sous réseau 255.255.0.0
- une passerelle par défaut 172.16.254.7

Les commandes suivantes doivent vous renseigner à ce sujet :

```
ip addr show (pour l'adresse IP et le masque de sous réseau)
```

```
ip route show (pour la route par défaut)
```

Vous pouvez également utiliser les commandes `ifconfig` et `route -n`

La résolution des noms doit se faire aussi correctement. Vérifiez-le, par exemple, avec la commande :

```
host jupiter.eme.org
```

qui doit vous retourner la réponse :

jupiter.eme.org has address 172.16.254.4

Une petite mise à jour...

Une bonne habitude à prendre est de vérifier s'il n'y a pas de mises à jour nécessaires. Pour cela, il suffit d'utiliser les commandes apt :

- `apt-get update` va synchroniser la liste locale des paquetages disponibles avec celle des serveurs indiquée dans le fichier `/etc/apt/sources.list`,
- `apt-get upgrade` va trouver automatiquement tous les paquetages à mettre à jour et le fera avec votre accord,
- si nécessaire, c'est à dire si la commande précédente vous a signalé que certains paquetages ne seront pas mis à jour, c'est que la structure de la distribution elle-même a été mise à jour, il faudra éventuellement (ce n'est pas obligatoire) utiliser encore la commande `apt-get dist-upgrade`.

Cette commande force une mise à jour complète du système lorsque c'est nécessaire. Sur une distribution dite « stable » (la « woody » que vous utilisez), ça arrive rarement.

Lorsque la version « Sarge » encore en phase de test, sera considérée comme stable, ce qui devrait se produire dans le courant de l'année, la commande `apt-get dist-upgrade` appliquée à une « woody » permettra de passer à « sarge ».

Mise en place pour la première partie

Une fois tout ceci vérifié, arrêtez votre machine, venez la brancher sur le HUB sans y connecter d'écran, il ne sera plus nécessaire. Le clavier, comme la souris, ne seront plus nécessaires non plus, si votre BIOS accepte de laisser démarrer le système sans ces accessoires.

Redémarrez votre machine. Le reste de la manipulation se fera depuis le poste Windows.

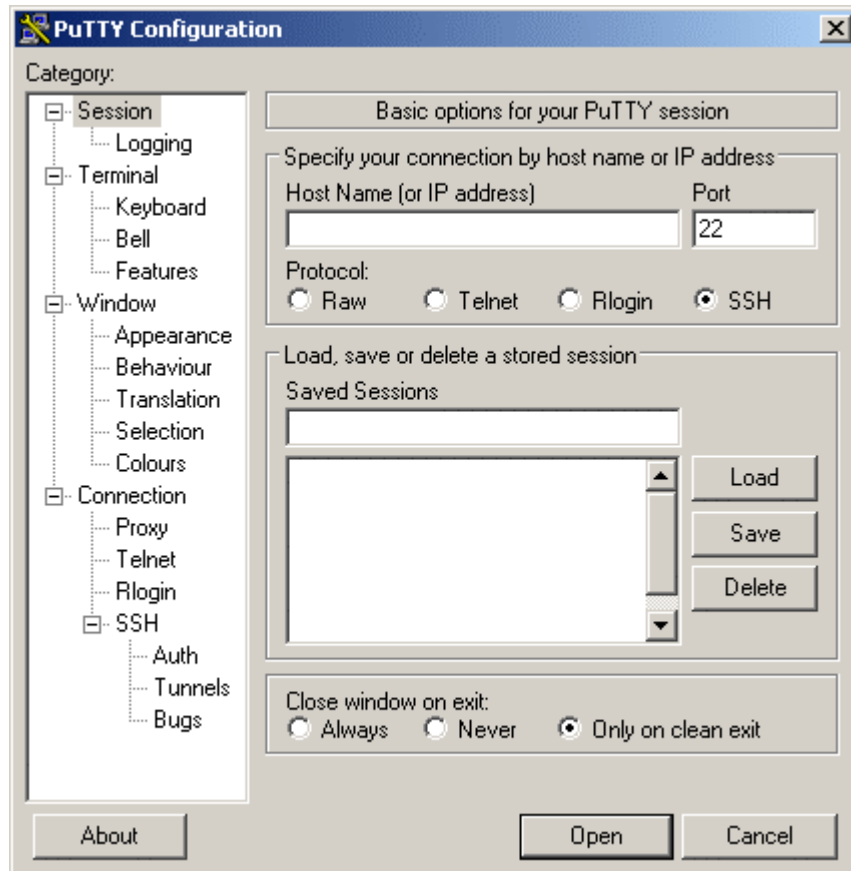
Sur votre machine Windows :

- ouvrez une session avec votre login habituel,
- allez chercher dans le répertoire partagé « `\\Orion\travaux_eleves\TS1\TP Info` » les deux applications « `putty` » et « `vnc-4.0-x86_win32_viewer` » et copiez les dans votre répertoire local « `c:\temp` ».

Première partie

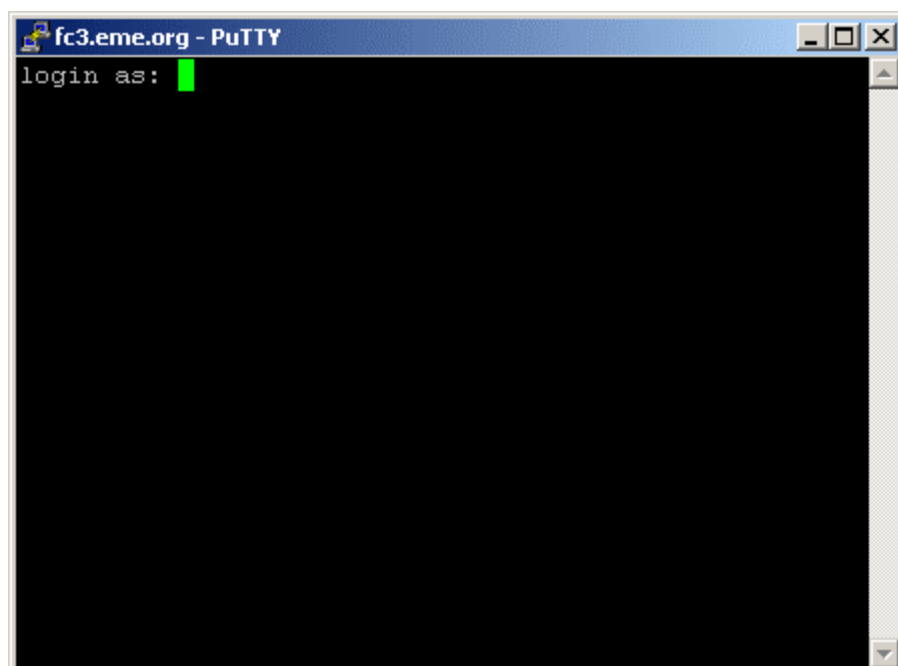
Putty

Putty est un utilitaire client ssh. Il va vous permettre d'ouvrir une session ssh sur votre machine Linux. Démarrez `putty.exe`, ce qui doit vous ouvrir une fenêtre comme celle-ci :



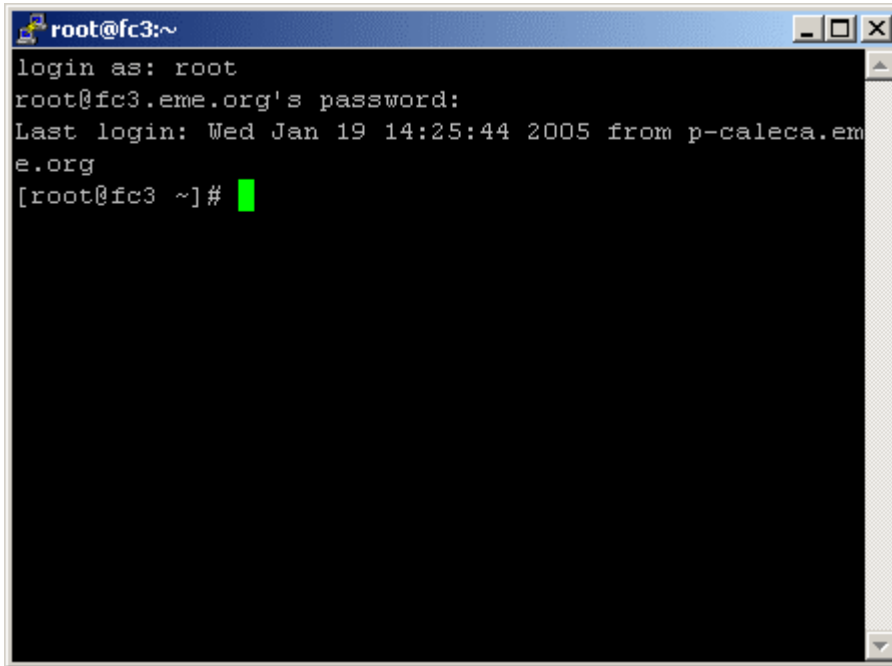
Dans « Host Name », indiquez l'adresse IP de votre machine Linux, puis cliquez sur le bouton « open ».

Ceci doit faire apparaître une fenêtre de cette forme :



Vous avez maintenant un terminal en mode texte (un « shell ») qui s'exécute sur votre machine Linux.

Ouvrez une session « root ». Vous êtes dans les mêmes conditions de travail que si vous aviez ouvert un « shell » localement sur votre machine Linux.



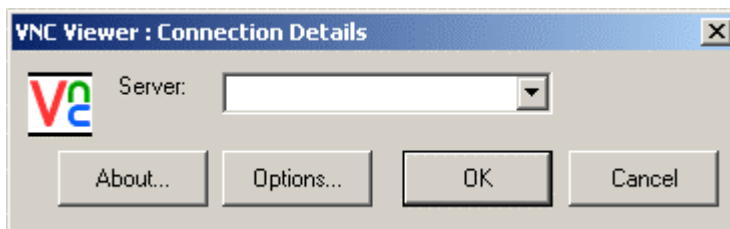
```
root@fc3:~  
login as: root  
root@fc3.eme.org's password:  
Last login: Wed Jan 19 14:25:44 2005 from p-caleca.em  
e.org  
[root@fc3 ~]#
```

Vous êtes maintenant en mesure de démarrer votre serveur vnc, avec la commande `vncserver` .

VncViewer

Le client `vncviewer` pour Windows permet d'ouvrir un terminal graphique attaché à une machine Linux sur un environnement Windows.

Dans Windows, démarrez maintenant votre client vnc :



Dans la zone « server » indiquez l'adresse IP de votre machine Linux, suivie de « :1 »

Cette information doit correspondre au numéro de « display » (relativement difficile à traduire en français), que `vncserver` a créé.

Une deuxième fenêtre s'ouvre, demandant un mot de passe, celui que vous avez indiqué la première fois que vous avez utilisé `vncserver` :



Si vous n'avez commis aucune erreur, vous devez vous retrouver avec une fenêtre complètement identique à celle que vous auriez eue si vous aviez démarré une session graphique localement sur votre machine Linux.

Je vous laisse le soin de trouver comment mettre cette fenêtre en plein écran.

A partir de maintenant, votre Windows ne vous sert plus que de support au protocole VNC, tout se passe comme si vous travaillez localement sur votre machine Linux.

Nous venons de mettre en place ce qui s'appelle un terminal graphique. Tout ce que vous faites dans ce terminal s'exécute sur votre machine Linux. Windows ne sert plus qu'à gérer l'affichage, le clavier et la souris.

Dans cet exemple, vous ne le constaterez pas, mais dans la pratique, une machine peu puissante qui supporte l'affichage graphique pourra servir de terminal pour une machine beaucoup plus puissante. C'est une technologie que l'on utilise assez souvent en informatique.

Si votre machine Linux était fortement pourvue en mémoire et dotée d'un (ou de plusieurs) processeur(s) rapide(s), il vous serait possible de créer sur la même machine Linux plusieurs « displays » en démarrant plusieurs fois vncserver. Vous obtiendriez alors les « displays » :1, :2, :3 etc. et plusieurs terminaux pourraient alors se connecter simultanément sur la même machine, chaque terminal utilisant un « display » différent, complètement indépendant des autres, qui donnerait l'impression à l'utilisateur d'être le seul à travailler sur la machine, alors qu'en réalité, il pourrait y avoir plusieurs utilisateurs simultanés.

Une telle manip mettrait clairement en évidence non seulement la notion de multi-tâches, mais aussi celle de multi-utilisateurs.

Mais il est possible de faire encore plus fort...

Dans le monde UNIX dont Linux fait partie, le mode graphique est assuré par un « serveur X » dans la grande majorité des cas. Vous utilisez l'implémentation libre du serveur X appelé Xfree86 sur la distribution Debian Woody. Ce serveur est maintenant de plus en plus remplacé par un autre serveur X, compatible, qui s'appelle X-org.

Qui dit « serveur » sous entend « client ».

Le principe est relativement simple. Lorsque vous démarrez votre serveur X, manuellement, ou automatiquement au démarrage de la machine, il crée un « display » d'index :0, destiné à travailler localement.

Votre écran local, votre clavier local, votre souris locale, vont être exploités par un client X local, qui se connecte localement au « display :0 ».

Votre serveur vnc va, éventuellement, créer des « displays » supplémentaires, comme on l'a vu plus haut, destinés à la connexion de clients X distants.

Mais une application qui tourne sous X (toute application en mode graphique) va dialoguer avec un « display » du serveur X (habituellement le « display :0 », ce qui permet à

l'utilisateur d'exploiter son application localement. Entendez par là que l'application tourne sur la même machine que celle qui génère l'écran graphique).

Dans le cas d'un terminal graphique, comme nous en avons mis un en œuvre avec vnc, bien que le terminal soit distant, tout revient à travailler localement.

Le système X permet de faire encore plus.

Il est en effet possible, par exemple, de démarrer une application sur une machine A, mais de produire l'affichage de la fenêtre de cette application (et seulement sa fenêtre) sur une autre machine B. L'application tourne sur B, mais l'interface de l'application est déportée sur la machine A.

Pour mettre en évidence cette possibilité, j'ai mis en place une autre machine Linux, qui fonctionne avec une distribution Fedora Core 3, pour bien montrer que tout ceci est indépendant de la distribution utilisée. Cette machine dispose du nom : « fc3.eme.org ». Vous pouvez vous assurer qu'elle est présente sur le réseau en faisant un :

```
ping fc2.eme.org
```

Voici ce qu'il va se passer...

Lorsque vous me signalerez que vous êtes prêts, je vais lancer sur fc2.eme.org une application, mais en lui indiquant qu'elle doit produire son interface sur le « display :1 » de votre machine Linux. Comme ce « display » correspond à votre terminal graphique, vous verrez l'application apparaître sur votre écran, et vous pourrez l'utiliser, comme si elle s'exécutait sur votre propre machine, alors qu'en réalité, elle s'exécutera sur fc3.eme.org.

Pour réaliser correctement cette manipulation, vous devez indiquer à votre machine qu'elle doit autoriser fc3 à utiliser votre serveur X, en employant dans une console shell les deux commandes suivantes :

- `xhost -` (pour n'autoriser aucune machine à utiliser votre affichage)
- `xhost + fc3.eme.org` (pour autoriser fc3 à le faire)

Lorsque vous aurez effectué cette opération, prévenez-moi pour que je puisse lancer sur fc3 l'application qui s'affichera sur votre « display ».

Observez et utilisez. Magique, non ?

Seconde partie

Le démineur, c'est bien, mais ce n'est pas très instructif.

Pour ce qui suit, comme nous allons avoir quelques ennuis de réseau, il vaudra beaucoup mieux que vous reconnectiez un écran, un clavier et une souris à votre machine linux.

Un Système GNU/Linux en état de fonctionner est en réalité un assemblage minutieux de briques logicielles qui sont en relations plus ou moins étroites entre elles.

Il y a certes le « kernel » qui constitue le cœur du système, qui est chargé en premier, mais il y a aussi une quantité plus ou moins importante, suivant la configuration, d'autres choses qui viennent s'ajouter pour constituer un système exploitable ;

Les modules

les modules sont des extensions du kernel, que l'on pourrait comparer aux « drivers » de Windows.

Le moyen le plus simple de connaître les modules qui sont chargés est d'utiliser la commande « `lsmod` » :

```
pxp-2001:~# lsmod
Module                Size  Used by    Not tainted
soundcore             3236   0 (autoclean)
apm                   9148   2 (autoclean)
keybdev              1664   0 (unused)
usbkbd                2848   0 (unused)
input                 3072   0 [keybdev usbkbd]
usb-uhci              20708  0 (unused)
usbcore               48032  0 [usbkbd usb-uhci]
```

Dans notre installation par défaut, il y en a relativement peu et certains sont même probablement inutiles (unused)

La plupart de ces modules est référencée dans un fichier de configuration spécifique : `/etc/modules` :

```
pxp-2001:~# cat /etc/modules
# /etc/modules: kernel modules to load at boot time.
#
# This file should contain the names of kernel modules that are
# to be loaded at boot time, one per line.  Comments begin with
# a "#", and everything on the line after them are ignored.

usb-uhci
input
usbkbd
keybdev
```

Décrire en détails le processus de démarrage d'un système GNU/Linux serait trop long et surtout trop compliqué. Sachez cependant qu'en principe, les modules signalés dans le fichier `/etc/modules` seront chargés dans l'ordre indiqué, au moment du démarrage, mais il peut y avoir d'autres modules chargés par d'autres moyens. A preuve, « `soundcore` », « `apm` » et tous les autres modules listés par « `modprobe` », et qui ne figurent pas dans « `/etc/modules` ».

Remarquez qu'il n'y a visiblement pas de module chargé pour la gestion de l'interface réseau. Pourtant, chaque interface nécessite une gestion spécifique, en fonction du modèle.

Il faut savoir que, lorsqu'on « compile » un noyau, il existe souvent la possibilité de choisir pour les routines de gestion de périphériques la façon dont on va s'en servir :

- il est possible de les compiler dans le noyau, ce qui est le cas du noyau `bf2.4` que nous utilisons actuellement. L'avantage est que le code exécutable est directement utilisable par le noyau, ce qui optimise les temps d'accès aux périphériques concernés,
- il est possible aussi de les compiler sous forme de modules. Ca donne un noyau plus petit, donc moins gourmand en mémoire, mais si le module n'est pas encore présent en mémoire (unused), le noyau devra d'abord le charger avant de pouvoir s'en servir. De plus, pour des raisons difficiles à expliquer simplement, l'accès aux routines dans les modules est plus long.

Le noyau par défaut que nous utilisons est compilé avec bon nombre de routines de gestion d'interfaces Ethernet, dont celle que nous utilisons. La manipulation qui suit va mettre ceci en évidence.

Nous prenons le parti de n'utiliser que des noyaux pré compilés, parce que ça présente quelques avantages :

- ils sont généralement bien compilés,
- ils sont faciles à mettre à jour avec les outils apt
- il est facile de se débarrasser des anciens noyaux qui ne servent plus.

Toutefois, si l'on souhaite optimiser son installation, la seule possibilité est alors de partir des sources du noyau et de le compiler soi-même, avec les options les plus appropriées à sa plate forme matérielle. Ce n'est pas très difficile, si l'on sait ce que l'on fait, mais c'est une procédure assez longue et il faut savoir aussi modifier le « boot loader » correctement pour utiliser ce nouveau noyau.

Certaines distributions, comme la Gentoo ou la Slackware, ne proposent que des paquetages source et nécessitent une compilation de tout le système. Sur une machine comme celle que nous utilisons pour les manips, avec les interfaces graphiques, ça représente environ une semaine de compilation « non stop ». A réserver, donc, aux puristes, spécialistes de GNU/Linux, qui disposent d'une machine rapide et de beaucoup de temps. Cependant, le résultat obtenu en termes d'efficacité est sans comparaisons avec une distribution pré compilée.

Remplacement du noyau initial.

Un bon moyen pour connaître la version du noyau en cours d'utilisation est d'utiliser la commande « `uname -a` » :

```
pxp-2001:~# uname -a
Linux pxp-2001 2.4.18-bf2.4 #1 Son Apr 14 09:53:28 CEST 2002 i686 unknown
```

C'est un noyau compilé pour poser le moins de problèmes possibles à l'installation, mais ce n'est pas un noyau optimisé pour la configuration dont nous disposons. Il n'utilise que les instructions génériques des processeurs 32 bits Intel, contient beaucoup de « drivers » qui nous sont inutiles...

Nous allons donc le remplacer par quelque chose de plus efficace.

Attention, la manipulation qui suit est très délicate et la moindre erreur aboutit à l'impossibilité de redémarrer la machine !!!

Suivez donc attentivement, en comprenant ce que vous faites...

D'abord, voyons ce que nous propose notre distribution :

```
pxp-2001:~# apt-cache search kernel-image-2.4.18
kernel-image-2.4.18-386 - Linux kernel image for version 2.4.18 on 386.
kernel-image-2.4.18-586tsc - Linux kernel image for version 2.4.18 on
Pentium-Classic.
kernel-image-2.4.18-686 - Linux kernel image 2.4.18 on
PPro/Celeron/PII/PIII/PIV.
kernel-image-2.4.18-686-smp - Linux kernel image 2.4.18 on
PPro/Celeron/PII/PIII/PIV SMP.
kernel-image-2.4.18-k6 - Linux kernel image for version 2.4.18 on AMD
K6/K6-II/K6-III
kernel-image-2.4.18-k7 - Linux kernel image for version 2.4.18 on AMD K7
kernel-headers-2.4.18-bf2.4 - Headers for Linux kernel version 2.4.18 (bf
variant) on 386
kernel-image-2.4.18-1-386 - Linux kernel image for version 2.4.18 on 386.
```

kernel-image-2.4.18-1-586tsc - Linux kernel image for version 2.4.18 on Pentium-Classic.
kernel-image-2.4.18-1-686 - Linux kernel image 2.4.18 on PPro/Celeron/PII/PIII/PIV.
kernel-image-2.4.18-1-686-smp - Linux kernel image 2.4.18 on PPro/Celeron/PII/PIII/PIV SMP.
kernel-image-2.4.18-1-k6 - Linux kernel image for version 2.4.18 on AMD K6/K6-II/K6-III
kernel-image-2.4.18-1-k7 - Linux kernel image for version 2.4.18 on AMD K7
kernel-image-2.4.18-bf2.4 - Linux kernel image for version 2.4.18 (bf variant) on 386.
pcmcia-modules-2.4.18-bf2.4 - PCMCIA Modules for Linux (kernel 2.4.18-bf2.4).

La commande « `apt-cache search` » permet d'effectuer une recherche dans la base de données des paquetages disponibles pour notre version.

Comme nous l'avons fait, nous avons recherché tous les paquetages dont le nom commence par « `kernel-image-2.4.6` ».

Celui qui est marqué en gras convient tout à fait à notre architecture, nous disposons d'un Pentium 3. Ce paquetage est compilé de manière à exploiter au mieux le jeu d'instruction de cette famille de processeurs, et n'inclut pas une trop grande quantité de « drivers » inutiles (et même parfois utiles) pour nous, nous allons d'ailleurs en faire l'expérience.

C'est parti :

```
pxp-2001:~# apt-get install kernel-image-2.4.18-686
Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed:
  ash cramfsprogs initrd-tools
The following NEW packages will be installed:
  ash cramfsprogs initrd-tools kernel-image-2.4.18-686
0 packages upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
Need to get 8802kB of archives. After unpacking 24.2MB will be used.
Do you want to continue? [Y/n]
```

Vous répondez Y

```
Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed:
  ash cramfsprogs initrd-tools
The following NEW packages will be installed:
  ash cramfsprogs initrd-tools kernel-image-2.4.18-686
0 packages upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
Need to get 8802kB of archives. After unpacking 24.2MB will be used.
Do you want to continue? [Y/n] Y
Get:1 ftp://mirdeb.eme.org stable/main ash 0.3.8-37 [71.6kB]
Get:2 ftp://mirdeb.eme.org stable/main cramfsprogs 1.1-3 [19.1kB]
Get:3 ftp://mirdeb.eme.org stable/main initrd-tools 0.1.32woody.3 [15.1kB]
Get:4 ftp://mirdeb.eme.org stable/main kernel-image-2.4.18-686 2.4.18-5
[8696kB]
Fetched 8802kB in 21s (400kB/s)
Preconfiguring packages ...
Sélection du paquet ash précédemment désélectionné.
```

```
(Lecture de la base de données... 31258 fichiers et répertoires déjà
installés.)
Dépaquetage de ash (à partir de ../archives/ash_0.3.8-37_i386.deb) ...
Sélection du paquet cramfsprogs précédemment désélectionné.
Dépaquetage de cramfsprogs (à partir de ../cramfsprogs_1.1-3_i386.deb) ...
Sélection du paquet initrd-tools précédemment désélectionné.
Dépaquetage de initrd-tools (à partir de ../initrd-
tools_0.1.32woody.3_all.deb) ...
Sélection du paquet kernel-image-2.4.18-686 précédemment désélectionné.
Dépaquetage de kernel-image-2.4.18-686 (à partir de ../kernel-image-
2.4.18-686_2.4.18-5_i386.deb) ...
```

```
You are attempting to install an initrd kernel image (version 2.4.18-686)
This will not work unless you have configured your boot loader to use
initrd. (An initrd image is a kernel image that expects to use an INITIAL
Ram Disk to mount a minimal root file system into RAM and use that for
booting).
```

```
As a reminder, in order to configure lilo, you need to
add an 'initrd=/initrd.img' to the image=/vmlinuz
stanza of your /etc/lilo.conf
```

```
I repeat, You need to configure your boot loader. If you have already done
so, and you wish to get rid of this message, please put
```

```
`do_initrd = Yes'
in /etc/kernel-img.conf. Note that this is optional, but if you do not,
you'll continue to see this message whenever you install a kernel
image using initrd.
```

```
Do you want to stop now? [Y/n]
```

Commencez par noter que les dépendances vous font installer quelques paquetages de plus, essayez ensuite de comprendre ce qui est dit dans la partie marquée en gras dans le message.

On vous signale que ce noyau ne peut fonctionner qu'avec un système de « ram disk » nommé « initrd », ce qui n'était pas le cas du bf24. En réalité, ce système crée une étape de boot supplémentaire, en chargeant d'abord un noyau minimal dans une zone de la RAM, qui sera temporairement vue comme un disque. En dire plus ne servirait à rien dans l'état actuel de nos connaissances. Il faut retenir de ceci que le « boot loader » (lilo dans notre cas) devra être configuré pour effectuer le chargement de ce « initrd », faute de quoi le noyau définitif ne pourra se charger et enverra un message de type « kernel panic », suivi d'un blocage irréversible du processus de chargement.

Enfin répondez non (n) à la question posée. Attention, ce n'est pas l'option par défaut.

```
Paramétrage de ash (0.3.8-37) ...
```

```
Paramétrage de cramfsprogs (1.1-3) ...
```

```
Paramétrage de initrd-tools (0.1.32woody.3) ...
```

```
Paramétrage de kernel-image-2.4.18-686 (2.4.18-5) ...
/boot/initrd.img does not exist. Installing from scratch, eh?
Or maybe you don't want a symbolic link here. Hmm? Lets See.
I notice that you do not have initrd.img symbolic
link. I can create one for you, and it shall be
updated by newer kernel image packages. This is
useful if you use a boot loader like lilo.
```

```
Do you want me to create a link from /boot/initrd.img-2.4.18-686 to
initrd.img?[Yn]
```

Répondez Y à la question

A new kernel image has been installed, and usually that means that some action has to be taken to make sure that the new kernel image is used next time the machine boots. Usually, this entails running a ``bootloader'' like SILO, loadlin, LILO, ELILO, QUIK, VMELILO, ZIPL, or booting from a floppy. (Some boot loader, like grub, for example, do not need to be run on each new image install, so please ignore this if you are using such a boot loader).

```
A new kernel image has been installed. at /boot/vmlinuz-2.4.18-686
(Size: 618kB)
```

```
Initial rootdisk image: /boot//initrd.img-2.4.18-686 (Size: )
```

Symbolic links, unless otherwise specified, can be found in /

LILO sets up your system to boot Linux directly from your hard disk, without the need for booting from a boot floppy.

WARNING

If you are keeping another operating system or another version of Linux on a separate disk partition, you should not have LILO install a boot block now. Wait until you read the LILO documentation. That is because installing a boot block now might make the other system un-bootable. If you only want to run this version of Linux, go ahead and install the boot block here. If it does not work, you can still boot this system from a boot floppy.

```
You already have a LILO configuration in /etc/lilo.conf
Install a boot block using the existing /etc/lilo.conf? [Yes]
```

Oui, faites "Entrée"

```
Testing lilo.conf ...
Testing successful.
Installing the partition boot sector...
Installation successful.
```

On croit que c'est bon, mais non... Il ne faut pas oublier le message d'alerte à propos du « initrd ».

Nous devons maintenant modifier le fichier /etc/lilo.conf pour prendre en compte cette alerte .

Au moyen de « mc », éditez ce fichier (/etc/lilo.conf). Repérez les lignes suivantes :

```
# Boot up Linux by default.
#
default=Linux

image=/vmlinuz
    label=Linux
    read-only
#    restricted
```

```
# alias=1

image=/vmlinuz.old
    label=LinuxOLD
    read-only
    optional
# restricted
# alias=2
```

Et modifiez pour que ça ressemble à ceci en ajoutant simplement la ligne en gras :

```
default=Linux

image=/vmlinuz
    initrd=/initrd.img
    label=Linux
    read-only
# restricted
# alias=1

image=/vmlinuz.old
    label=LinuxOLD
    read-only
    optional
# restricted
# alias=2
```

N'oubliez pas d'enregistrer la modification.

Le fichier de configuration de lilo est maintenant correct, mais ça ne suffit pas, il faut reconstruire le MBR en fonction de cette nouvelle configuration en utilisant la commande « lilo -v » :

```
pxp-2001:~# lilo -v
LILLO version 22.2, Copyright (C) 1992-1998 Werner Almesberger
Development beyond version 21 Copyright (C) 1999-2001 John Coffman
Released 05-Feb-2002 and compiled at 20:57:26 on Apr 13 2002.
MAX_IMAGES = 27
```

```
Reading boot sector from /dev/hda
Merging with /boot/boot-menu.b
Warning: Int 0x13 function 8 and function 0x48 return different
head/sector geometries for BIOS drive 0x80
Boot image: /vmlinuz -> boot/vmlinuz-2.4.18-686
Mapping RAM disk /initrd.img -> /boot/initrd.img-2.4.18-686
Added Linux *
```

```
Boot image: /vmlinuz.old -> boot/vmlinuz-2.4.18-bf2.4
Added LinuxOLD
```

```
/boot/boot.0300 exists - no backup copy made.
```

Writing boot sector.

Attention, si vous ne voyez pas la ligne « Writing boot sector », c'est que vous avez fait une erreur dans votre lilo.conf ! Trouvez-la et corrigez-la, puis recommencez lilo -v

Voilà. normalement, en rebootant, nous devrions nous retrouver avec le nouveau noyau :

```
pxp-2001:~# reboot
```

Ca doit fonctionner.

Quelques remarques tout de même...

- Les paquetages pré compilés proposent, en plus des binaires à installer, deux scripts qui s'exécutent :
 - avant la mise en place des binaires (pre install) pour éventuellement opérer des manœuvres nécessaires à la préparation de l'installation,
 - après la mise en place des binaires (post install) le plus souvent pour configurer correctement les binaires installés.
- Ici, ces scripts ont, entre autres :
 - créé des liens symboliques vers le « initrd » et vers le « kernel »
 - modifié (mais pas complètement, on se demande d'ailleurs pourquoi), le fichier de configuration de LILO.

Si nous avons décidé de compiler nous même notre noyau, nous aurions du réaliser manuellement ces opérations.

- Vous avez pu constater dans votre « lilo.conf », qu'il y a deux paragraphes qui se ressemblent beaucoup. En réalité, les scripts « post install » de notre nouveau kernel ont créé un « multi-boot », qui nous permet, si nécessaire, de rebooter sur l'ancien noyau bf24. Pour le constater, il vous suffit d'utiliser la commande « reboot » et, lorsque LILLO : s'affichera à l'écran, appuyez très rapidement (vous avez 5 secondes) sur la touche « shift » (celle qui permet d'obtenir temporairement les majuscules). Vous verrez alors apparaître un écran de choix pour le multi-boot.
- Nous avons installé un nouveau noyau pré compilé, mais que ceci soit clair : c'est le même noyau que le précédent, il est juste compilé avec des options différentes. Lorsque nous passerons à un noyau de type 2.6, si nous mettons à jour vers « sarge », bien que la procédure soit la même, nous changerons effectivement de noyau (version 2.6 à la place de notre 2.4 actuelle). La procédure sera la même, mais nécessitera une mise à jour en profondeur du système, il y a beaucoup de choses qui changent entre les 2.4 et les 2.6.

Et le réseau ?

Vous n'avez plus de réseau !

Pour le vérifier, utilisez par exemple la commande « ifconfig » :

```
pxp-2001:~# ifconfig
lo                Lien encap:Boucle locale
                  inet adr:127.0.0.1  Masque:255.0.0.0
                  UP LOOPBACK RUNNING  MTU:16436  Metric:1
                  RX packets:10 errors:0 dropped:0 overruns:0 frame:0
                  TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 lg file transmission:0
                  RX bytes:500 (500.0 b)  TX bytes:500 (500.0 b)
```

L'interface eth0 n'est bel et bien pas présente.

Ce noyau n'est pas compilé avec les interfaces réseau. Il va nous falloir trouver le bon module et le charger.

Les modules relatifs aux interfaces réseau sont dans le répertoire :

lib/modules/2.4.18-686/kernel/drivers/net

Si vous n'avez aucune idée du module à charger, vous pouvez les essayer un par un, jusqu'à tomber sur le bon, mais c'est mieux de savoir, donc je vous dis lequel c'est : il s'agit de 8139too.o (vous n'avez aucun moyen de le savoir autrement qu'en regardant sur votre carte la référence du circuit contrôleur Ethernet, qui est un RTL8139).

Pour installer un module, il suffit d'utiliser la commande « modprobe » :

```
pxp-2001 :# modprobe 8139too
```

S'il n'y a pas d'erreur annoncée, c'est que le module s'est bien installé.

Maintenant, il faut recharger la configuration du réseau :

```
pxp-2001 :# /etc/init.d/networking restart
```

Il peut y avoir des alertes qui s'affichent, sans gravité. Vérifions maintenant que nous avons bien notre interface eth0 :

```
pxp-2001:~# ifconfig
```

```
eth0      Lien encap:Ethernet  HWaddr 00:00:E8:8E:CB:72
          inet adr:172.16.51.1  Bcast:172.16.255.255  Masque:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:158 errors:0 dropped:0 overruns:0 frame:0
          TX packets:49 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:100
          RX bytes:25079 (24.4 KiB)  TX bytes:6522 (6.3 KiB)
          Interruption:10 Adresse de base:0xe000
```

```
lo        Lien encap:Boucle locale
          inet adr:127.0.0.1  Masque:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:10 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:0
          RX bytes:500 (500.0 b)  TX bytes:500 (500.0 b)
```

Vous pouvez aussi utiliser la commande plus « moderne » :

```
ip addr list
```

Et voilà le travail.

Au passage, vérifions un peu les modules qui sont chargés :

```
pxp-2001:~# lsmod
```

Module	Size	Used by	Not tainted
8139too	13632	1	
mii	1088	0 [8139too]	
soundcore	3556	0 (autoclean)	
apm	8892	2 (autoclean)	
keybdev	1664	0 (unused)	
usbkbd	2912	0 (unused)	
input	3328	0 [keybdev usbkbd]	
usb-uhci	21028	0 (unused)	
usbcore	48192	0 [usbkbd usb-uhci]	
rtc	5368	0 (autoclean)	
reiserfs	151552	2 (autoclean)	
isofs	24224	0 (autoclean)	
vfat	9276	0 (autoclean)	

```
fat                29080    0 (autoclean) [vfat]
ext2               30400    0 (autoclean)
ide-disk           6592     3 (autoclean)
ide-probe-mod      7968     0 (autoclean)
ide-mod            129420   3 (autoclean) [ide-disk ide-probe-mod]
ext3               56544    0 (autoclean)
jbd                34968    0 (autoclean) [ext3]
unix               13316   76 (autoclean)
```

Nous retrouvons bien 8139too plus beaucoup d'autres, dont « reiserfs », le système de gestion de fichiers, qui est cette fois-ci configuré par un module.

Pourtant, si nous retournons voir le contenu de /etc/modules, il n'y a rien eu de changé. Il faut d'ailleurs éditer ce fichier pour ajouter le module 8139too sinon, au prochain redémarrage, nous n'aurons de nouveau plus de réseau.

Avec mc, ajoutons donc cette ligne, ce qui fera que maintenant, /etc/modules doit ressembler à ceci :

```
# /etc/modules: kernel modules to load at boot time.
#
# This file should contain the names of kernel modules that are
# to be loaded at boot time, one per line.  Comments begin with
# a "#", and everything on the line after them are ignored.
```

8139too

```
usb-uhci
input
usbkbd
keybdev
```

N'oubliez toujours pas d'enregistrer la modification. Quittez mc et rebootez avec la commande du même nom.

Votre interface réseau est automatiquement opérationnelle.

Bien sûr, en sachant exactement ce qu'il fallait faire dès le début de la manip, nous aurions pu réaliser tout ça à distance. Il suffisait, avant de rebooter, juste après l'installation du nouveau kernel, d'ajouter la bonne ligne dans /etc/modules, et nous n'aurions pas eu l'ennui de ne plus avoir de réseau.

Les « services »

Voyons maintenant tout ce qui tourne en permanence autour du noyau, qui sont de simples applications, mais qui sont nécessaires, suivant la configuration souhaitée.

Le moyen le plus simple est d'utiliser la commande « ps aux » :

```
pxp-2001:~# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1  1272   484 ?        S    15:35   0:00 init [2]
root         2  0.0  0.0      0     0 ?        SW   15:35   0:00 [keventd]
root         3  0.0  0.0      0     0 ?        SWN15:35 0:00 [ksoftirqd_CPU0]
root         4  0.0  0.0      0     0 ?        SW   15:35   0:00 [kswapd]
root         5  0.0  0.0      0     0 ?        SW   15:35   0:00 [bdflush]
root         6  0.0  0.0      0     0 ?        SW   15:35   0:00 [kupdated]
root        37  0.0  0.0      0     0 ?        SW   15:35   0:00 [kreiserfsd]
root         78  0.0  0.0      0     0 ?        SW   15:35   0:00 [khubd]
root        134  0.0  0.0      0     0 ?        SW   15:35   0:00 [eth0]
```

```

root      199  0.0  0.2  1344  596 ?      S   15:35   0:00 /sbin/syslogd
root      202  0.0  0.4  1904 1088 ?      S   15:35   0:00 /sbin/klogd
root      207  0.0  0.0     0    0 ?      SW  15:35   0:00 [kapmd]
root      209  0.0  0.1  1268  488 ?      S   15:35   0:00 /usr/sbin/apmd -
root      217  0.0  0.1  1292  508 ?      S   15:35   0:00 /usr/sbin/inetd
root      227  0.0  0.4  2784 1208 ?      S   15:35   0:00 /usr/sbin/sshd
root      233  0.0  1.3  4480 3392 ?      S   15:35   0:00 /usr/bin/X11/xf86
daemon    234  0.0  0.2  1384  580 ?      S   15:35   0:00 /usr/sbin/atd
root      237  0.0  0.2  1652  684 ?      S   15:35   0:00 /usr/sbin/cron
root      244  0.0  0.2  2516  684 ?      S   15:36   0:00 /usr/bin/kdm
root      249  0.0  0.1  1252  468 tty1 S   15:36   0:00 /sbin/getty
root      250  0.0  0.1  1252  468 tty2 S   15:36   0:00 /sbin/getty 38400
root      251  0.0  0.1  1252  468 tty3 S   15:36   0:00 /sbin/getty 38400
root      252  0.0  0.1  1252  468 tty4 S   15:36   0:00 /sbin/getty 38400
root      253  0.0  0.1  1252  468 tty5 S   15:36   0:00 /sbin/getty 38400
root      255  0.0  0.1  1252  468 tty6 S   15:36   0:00 /sbin/getty 38400
root      256  0.1  3.6 18944 9248 ?      S<  15:36   0:01 /usr/X11R6/bin/X -
root      257  0.0  0.3  2520  804 ?      S   15:36   0:00 -:0
root      260  0.0  2.4 13112 6384 ?      S   15:36   0:00 /usr/bin/kdm_greet
root      267  0.0  0.6  5724 1772 ?      S   15:37   0:00 /usr/sbin/sshd
root      269  0.0  0.5  2480 1296 pts/0 S   15:37   0:00 -bash
root      279  0.0  0.5  3288 1412 pts/0 R   15:56   0:00 ps aux

```

Il n'est bien entendu pas question de détailler toutes ces lignes, mais prenons en exemple deux de ces lignes :

```

root      227  0.0  0.4  2784 1208 ?      S   15:35   0:00 /usr/sbin/sshd
root      244  0.0  0.2  2516  684 ?      S   15:36   0:00 /usr/bin/kdm

```

La première correspond à un serveur, le serveur ssh, celui qui permet de se connecter à distance sur la machine, nous l'avons utilisé en première partie de cette manip.

La seconde correspond à une tâche de fond, qui permet de démarrer le serveur X et de le redémarrer en cas de problème ou simplement en cas de changement de session utilisateur

Entre autres particularités, tout « service », ou « daemon » s'exécute sous le nom d'un utilisateur enregistré (généralement root, mais ce n'est pas une obligation, bien au contraire), c'est ce qui est indiqué dans la première colonne.

Une autre caractéristique est le PID (Process IDentificator), c'est le nombre indiqué en seconde colonne. Repérez sur votre machine le PID du daemon kdm (244 sur mon exemple, mais pas forcément sur votre machine) et « tuez » ce daemon avec la commande « kill » :

```
pxp-2001:~# kill 244
```

il est mort. Votre écran est passé en mode texte.

Pour revenir à un mode graphique, utilisez, par exemple, la commande : « /etc/init.d/kdm start » :

```
pxp-2001:~# /etc/init.d/kdm start
Starting K Desktop Manager: done.
```

Votre écran est à nouveau graphique. Vous avez créé un nouveau daemon kdm, mais il n'aura pas le même PID que le précédent.

Le système V

Depuis quelques temps maintenant, les distributions Linux utilisent un système d'initialisation appelé « SysVinit ». C'est un processus complexe que nous n'allons pas détailler, mais en quelques mots, voici comment ça fonctionne.

Sur une Debian (sur d'autres distributions, ça peut être légèrement différent, mais le principe reste le même), il existe un répertoire : « /etc/init.d », qui contient des scripts normalisés. Ces scripts connaissent tous au moins les trois paramètres « start », « stop » et « restart » (d'où la commande « /etc/init.d/kdm start »).

Par ailleurs, il existe 7 répertoires, nommés /etc/rc.n, avec « n » variant de 0 à 7. Chacun de ces répertoires contient des liens symboliques (proches des raccourcis de Windows) vers des scripts de /etc/init.d, préfixés par un S et un index sur deux chiffres, ou un K suivi de 2 chiffres.

Lorsqu'il y a un S (start) il faut démarrer le service, lorsqu'il y a un K, il faut le tuer. L'index sert à déterminer l'ordre des opérations.

Par exemple, dans /etc/rc2.d, on trouve le lien S99kdm, ce qui veut dire qu'il faut démarrer le daemon kdm en dernier (équivalent à la ligne de commande /etc/init.d/kdm start).

Pourquoi y a-t-il 7 répertoires différents ?

- rc0.d est utilisé lorsqu'on arrête la machine,
- rc6 est utilisé lorsqu'on reboote la machine
- rc1 est un peu spécial, il sert en démarrage mono-utilisateur (pour des cas de maintenance difficile)
- rc2 à rc5 proposent différentes configurations de démarrage.

La configuration de démarrage par défaut est indiquée (fort peu clairement) dans le fichier /etc/inittab. Sur Debian, le démarrage par défaut se fait sur rc2.

Il est possible d'outrepasser ce réglage par défaut au moment du boot, si l'on sait exactement quoi taper et qu'on sait le faire très vite, au moment où l'on voit apparaître « LILO : » à l'écran.

Il est également possible de construire un multiboot qui présentera plusieurs options de démarrage dans un menu, correspondant à différents rc.

Sur Windows, bien que ce ne soit pas très connu, il est également possible de construire plusieurs profils de démarrage.

Mais c'est quoi, un serveur ?

Une définition

Un serveur, c'est un système qu'on interroge et dont on attend une réponse. Un serveur Web, un serveur FTP, un serveur de messagerie, un serveur SSH...

Celui qui pose la question, c'est le client. Un client pose une question à un serveur et le serveur répond à la question du client.

Un peu de réflexion...

Un client prend l'initiative de poser une question à un serveur, quand il veut. Il a juste besoin de savoir :

- à qui s'adresser (adresse IP),
- comment formuler sa question (protocole de communication).

Un serveur ne pose jamais la première question (il peut éventuellement en poser au client, mais une fois que le dialogue a été initié par le client). En revanche, comme il ne peut pas savoir quand un client va l'interroger, il doit rester tout le temps à l'écoute. En informatique, le seul moyen de rester tout le temps à l'écoute de quelque chose est de créer une application qui va fonctionner en permanence pour réaliser cette tâche (sshd, par exemple, pour les connexions ssh).

Sur une même machine (même adresse IP), il n'est pas rare de vouloir placer plusieurs serveurs différents, nous le ferons probablement dans des manip ultérieures. Sur une machine comme jupiter.eme.org, il y a plusieurs serveurs installés parmi lesquels :

- un serveur web,
- un serveur DNS
- un serveur DHCP

Et d'autres encore. Pourtant, cette machine ne dispose que d'une seule adresse IP. Comment faire alors, pour que la requête d'un client DNS soit bien aiguillée sur le serveur DNS, alors que dans le même temps, une requête HTTP (web) émise par un autre client, va arriver sur la même adresse IP ?

Il faut bien entendu un mécanisme d'aiguillage, qui est mis en place par la notion de ports de communication.

Un port est un nombre entier, non signé, sur 16 bits (65536 ports possibles), et un client http, en plus de connaître l'adresse IP du serveur qu'il veut interroger, doit également connaître sur quel port ce serveur écoute.

En étant déjà à la page 18 de ce sujet de manip, le reste vous sera dit oralement.