Cahiers de l'Admin

Collection dirigée par Nat Makarévitch

BSD

Emmanuel **Dreyfus**

Avec la contribution
d'Ollivier **Robert** et d'Olivier **Tharan**

2e édition

EYROLLES

Dans la collection

Les Cahiers de l'Admin

dirigée par Nat Makarévitch

Mac OS X Server

Jacques FOUCRY – N°11192, 2003.

On verra dans ce cahier que Mac OS X Server version 10.2 (Jaguar) facilite considérablement la vie de l'administrateur : outre ses fonctions d'administration et d'automatisation évoluées, il prend en charge tous types de clients (Mac OS, UNIX/Linux, Windows) et offre en standard la panoplie désormais indispensable d'outils Open Source MySQL, PHP, Samba, NFS, FTP, CUPS…

Sécuriser un réseau Linux 2e édition

Bernard BOUTHERIN, Benoit DELAUNAY - N°11445, 2004.

À travers une étude de cas générique mettant en scène un réseau d'entreprise, l'administrateur apprendra à améliorer l'architecture et la protection de ses systèmes connectés, notamment contre les intrusions, dénis de service et autres attaques : filtrage des flux, sécurisation par chiffrement avec SSL et (Open) SSH, surveillance quotidienne… On utilisera des outils Linux libres, réputés pour leur efficacité.

BSD 2e édition

Emmanuel DREYFUS – N°11244, 2003.

Ce cahier révèle les dessous d'UNIX et détaille toutes les opérations d'administration UNIX/BSD : gestion des comptes, initialisation de la machine, configuration des serveurs web, DNS et de messagerie, filtrage de paquets… Autant de connaissances réutilisables sous d'autres systèmes UNIX, et en particulier Linux.

Chez le même éditeur

J.-L. BÉNARD, L. BOSSAVIT, R. MÉDINA, D. WILLIAM

L'Extreme Programming

N. 11051, 2002, 350 pages

V. STANFIELD & R..W. SMITH

Guide de l'administrateur Linux

N°11263, 2003, 654 pages.

C. AULDS.

Apache 2.0 Guide de l'administrateur Linux.

N°11264, 2003, 582 pages.

G. MACQUET – **Sendmail**

N°11262, 2003, 293 pages.

C. HUNT. – **Serveurs réseau Linux.**

N°11229, 2003, 650 pages.

R. RUSSELL et al. – **Stratégies anti-hackers**

N°11138, 2e édition 2002, 754 pages.

Dans la collection

Les Cahiers du programmeur

Mac OS X

Gestionnaire de photos avec Cocoa, REALbasic et WebObjects.

Alexandre Carlhian, Jacques Foucry, Jean-Philippe Lecaille, Jayce Piel – avec la collaboration d'Olivier Gutknecht – N°11192, 2003.

Réalisez un gestionnaire de photos consultable via le Web avec Cocoa et Objective-C, REALbasic et WebObjects.

PHP 5

Application de chat avec PHP 5 et XML

Stéphane Mariel – N°11234, 2004.

De la conception à l'exploitation, on créera une application de discussion en ligne en PHP 5 respectant les méthodes éprouvées du développement web : architecture MVC, conception modulaire avec les interfaces, sessions, gestion d'erreurs et exceptions, échanges XML et transformations avec DOM, XPath et SimpleXML.

PHP (2)

Ateliers Web professionnels avec PHP/MySQL et JavaScript.

Philippe CHALEAT et Daniel CHARNAY – N°11089, 2002.

En une douzaine d'ateliers pratiques, allant de la conception d'aides multi-fenêtrées en JavaScript à la réalisation de services Web, en passant par les templates PHP et les annuaires LDAP, on verra qu'autour de formulaires HTML, on peut sans mal réaliser des applications légères ergonomiques et performantes.

PostgreSQL

Services Web professionnels avec PostgreSQL et PHP/XML.

Stéphane MARIEL – N°11166, 2002.

Ce cahier montre comment réaliser simplement des services Web avec PostgreSQL, PHP et XML. Le développeur apprendra à modéliser sa base, tirer parti de la richesse de PostgreSQL (transactions, procédures stockées, types de données évolués…), optimiser ses performances et en automatiser l'administration, sans oublier la réalisation d'un affichage dynamique avec XSLT.

Dans la collection

Accès libre

Débuter sous Linux.

S. Blondeel, H. Singodiwirjo. – N°11349, 2004, 328 pages.

Cet ouvrage guidera des utilisateurs motivés, qu'ils aient ou non utilisé un système MS-Windows, vers la connaissance, l'utilisation et l'administration du système libre et gratuit Linux, qui offre, outre sa puissance, les indispensables de tout poste de travail : traitements de texte, tableurs, mail, navigation Web, messagerie instantanée, jeux, multimédia.

OpenOffice.org efficace.

S. Gautier, C. Hardy, F. Labbe, M. Pinquier.

N°11348, 2004, 350 pages.

OpenOffice.org, suite bureautique gratuite tournant sous Windows, Linux et Mac OS X, inclut tous les modules habituels : traitement de texte, tableur de calcul, présentation, dessin, formules… Écrit par les chefs de file du projet français OpenOffice.org, cet ouvrage montre comment optimiser son environnement de travail et l'utilisation de chaque module d'OOo, comment s'interfacer avec des bases de données telle MySQL, et offre enfin un précis de migration.

Réussir un site Web d'association...

avec des outils gratuits.

A.-L. Quatravaux, D. Quatravaux. – N°11350, 2004, 280 pages.

Souvent peu dotée en moyens, une association doit gérer ses adhérents et membres, faciliter l'organisation du travail entre eux, être visible et soigner son image. Depuis le choix de l'hébergement jusqu'au référencement, en passant par la personnalisation graphique sous SPIP, la configuration d'un serveur d'e-mailing et la création de listes de diffusion, ce livre explique comment gagner un temps précieux en confiant ces tâches à des outils adéquats et gratuits.

Emmanuel **Dreyfus**

Cahiers de l'Admin BSD

2e édition

Les dessous d'Unix

Collection dirigée par Nat **Makarévitch**

Avec la contribution d'Ollivier **Robert**, Olivier **Tharan**
Florence **Henry** et Sébastien **Blondeel**

EYROLLES

Remerciements

Cet ouvrage ne serait pas ce qu'il est sans le travail des relecteurs. Un grand merci donc à Manuel Bouyer, Emmanuel Eisenstaedt, Gérard Henry, et Xavier Humbert pour les remarques et suggestions qu'ils ont pu apporter à ce livre. Merci aussi à Sébastien Blondeel, Ollivier Robert, et Olivier Tharan, qui furent les acteurs de la grande relecture finale. S'il reste des coquilles, c'est de leur faute !

Ce livre est écrit en DocBook XML avec l'éditeur **`vi`**. Il est difficile d'imaginer le résultat final lorsque l'on travaille avec de tels outils, mais les formats XML ont pour principal avantage d'être faciles à transformer.

Le document DocBook est ainsi transformé en un document LaTeX grâce aux feuilles de styles XSL du projet DB2LaTeX, de Ramon Casellas et James Devenish, dont il faut au passage saluer le remarquable travail. Le fichier LaTeX obtenu est ensuite traité par une feuille de style pour produire une mise en page proche de la maquette de la collection mise au point par l'éditeur. Je dois cette feuille de style à Florence Henry, que je remercie chaleureusement.

Avant-propos

Unix a la peau dure. Alors que ses versions pour les stations de travail des grands constructeurs (IBM, HP, SGI, Sun) sont sur le déclin, on le voit ressurgir en force à travers MacOS X et les Unix libres comme GNU/Linux et les BSD.

Unix a bien des qualités : fiable, performant, flexible (il a survécu à plus de 30 ans d'évolution informatique), et gratuit avec les Unix libres. Hélas, Unix a un défaut : c'est un système complexe.

Mais cette complexité est contrebalancée par une autre de ses qualités : Unix est transparent. On peut comprendre pourquoi et comment les choses se passent sous Unix, ce qui n'est pas forcément le cas sur d'autres systèmes. Ceux qui ont déjà écumé la base de connaissances de Microsoft à la recherche du nom de la clef de base de registres à changer pour modifier un comportement donné apprécieront.

Sa complexité est donc maîtrisable, au prix d'un investissement personnel : la pratique. Ce livre jette les bases pour le lecteur disposé à y consacrer du temps.

Nous verrons les bases d'administration système Unix dans le cas des BSD. Le chapitre 1 présente la famille Unix et la place que les systèmes BSD y occupent. On y expliquera également pourquoi les BSD sont de bons systèmes d'apprentissage et de bons choix en production. Le chapitre 2 présente l'étude de cas familière aux habitués de la collection.

Nous enchaînerons sur les bases de l'utilisation d'une machine Unix au chapitre 3, pour continuer avec l'installation au chapitre 4. Cet ordre peut surprendre, mais il faudra parfois connaître les bases pour pouvoir installer.

Les chapitres 5 à 8 traitent d'administration système Unix. Nous exposons d'abord la procédure de démarrage de la machine, bien utile à connaître pour résoudre les problèmes en cas de panne. Vient ensuite la présentation de quelques tâches quotidiennes sur une machine Unix : gestion des utilisateurs, des groupes, et des permissions. Enfin, nous abordons les tâches de configuration les plus fréquentes sur une station Unix : le réseau, l'interface graphique, l'intégration dans un réseau de machines Unix.

Le chapitre 10 traite de la configuration de systèmes BSD en pare-feu, domaines où ils sont très prisés. La compréhension de ce type de configuration exige des connaissances assez approfondies sur les réseaux locaux et TCP/IP, thèmes qui font l'objet d'une rapide introduction au chapitre 9 .

Le chapitre 11 explique l'installation de logiciels de tierces parties, par recompilation manuelle ou via un système de paquetages. Sa lecture permettra d'installer les nombreux logiciels libres disponibles gratuitement.

VOCABULAIRE **Logiciel tierce partie**

Ce terme désigne les logiciels non fournis dans le système BSD mais écrits par des tiers. Des milliers de logiciels libres et propriétaires sont ainsi disponibles.

Retour sur un plan plus opérationnel : le chapitre 12 détaille la configuration de services classiques de l'Internet ou de l'Intranet : le Web, le DNS, et la messagerie.

Enfin, le chapitre 13 analyse les catastrophes : il passe en revue différentes difficultés qui peuvent se présenter sur des systèmes Unix, et propose quelques solutions. Il traite des mises à jour de sécurité, des sauvegardes, et de la surveillance du système.

Ce que vous ne trouverez pas dans cet ouvrage

Ce volume ne comporte pas de fichier de configuration prêt à l'emploi ni de recette universelle. L'apprentissage de l'administration Unix demande des efforts et de la pratique : c'est en écrivant des fichiers de configuration et en mettant au point ses propres solutions que l'on devient un administrateur Unix confirmé. Ce livre n'est pas un livre de recettes ; il tente plutôt d'aider à comprendre.

On ne trouvera pas non plus ici une compilation de la documentation existante, ni des modes d'emploi « pas à pas » de mise en place d'une fonction précise. Cet ouvrage renverra souvent aux sources d'information en ligne, sans s'y substituer, pour éviter une obsolescence rapide : le Web est toujours plus à jour que les livres.

Ce que vous trouverez dans cet ouvrage

Ce livre présente des informations sur l'administration et le fonctionnement des systèmes Unix en général, et sur les systèmes BSD en particulier. Lorsque cela

est possible, il mettra en exergue les différences entre Unix distincts pour que le lecteur acquière des connaissances réutilisables d'un système à l'autre.

Certains chapitres valent donc pour la plupart des Unix. D'autres, spécifiques aux systèmes BSD, traitent de domaines où les systèmes Unix diffèrent beaucoup entre eux. Les chapitres sur l'installation, la configuration d'une station, et la mise en place d'un pare-feu sont ainsi largement orientés BSD.

Des mises en gardes évoquent les chausse-trapes qui piègent fréquemment les débutants sous Unix : on proposera des solutions pour les éviter ou contourner.

À qui s'adresse cet ouvrage ?

Ce livre s'adresse à qui désire acquérir de solides bases d'administration système Unix, dans le cadre d'une utilisation domestique comme professionnelle. Les études de cas proposées vont de la mise en place d'un pare-feu personnel à l'intégration de serveurs d'entreprise pour des réseaux éventuellement hétérogènes. Beaucoup d'autres choses sont possibles, moyennant de prendre le temps de se plonger dans la configuration de la machine.

Ce manuel cible des débutants sous Unix, dotés d'une bonne culture informatique ; il se veut le compagnon d'un apprentissage pratique. Le lecteur prêt à installer un système BSD sur une machine et à passer du temps à le manipuler pourra réellement rompre l'os et profiter de la substantifique moelle.

Quiconque se sera reconnu dans cette description pourra sans tarder se précipiter au chapitre 1 !

Conventions typographiques

Un certain nombre de conventions typographiques ont été adoptées pour faciliter la lecture de ce livre et permettre l'identification rapide de certains éléments.
Les noms ou expressions en langue étrangère apparaissent comme ceci : *if it is not broken, do not fix it*.
Les noms de commandes apparaissent ainsi : **`disklabel`**, et les options qui leur sont associées ainsi : **`-e -I`**. Les noms de fichiers sont imprimés ainsi : /etc/passwd.
Les extraits de fichiers sont rendus dans le style suivant :

```
if [ "$1" = autoboot ]; then
        autoboot=yes
        rc_fast=yes
fi
```

Et les sessions interactives sont rendues ainsi :

```
$ grep manu /etc/passwd|awk '{print $3}'
manu:*:500:500:Emmanuel Dreyfus:/home/manu:/bin/ksh
$ grep manu /etc/passwd|awk -F: '{print $3}'
500
```

Dans ces sections, et dans le reste du livre, ce qui doit être saisi par l'utilisateur apparaît ainsi : **`shutdown -r now`**, et ce qui est affiché par la machine apparaît ainsi : `ksh: help: not found`.

Table des matières

1

Unix et BSD

La famille Unix est très nombreuse. Essayons d'en démêler les branches pour comprendre ce qui lie des systèmes tels que les BSD, Linux, et les Unix constructeurs.

SOMMAIRE

MOTS-CLEFS

- Unix
- Systèmes BSD

Un peu de généalogie

Vous connaissez probablement déjà un peu Unix ou Linux, car vous avez choisi de feuilleter ce livre. Linux fait beaucoup parler de lui ces derniers temps ; on peut même lire dans certains journaux qu'il saura délivrer la bureautique du joug microsoftien. C'est tout le mal qu'on lui souhaite. Unix reste moins connu du grand public, qui a parfois du mal à se figurer en quoi il consiste exactement, et quels sont les liens qui le rattachent à Linux. Quant aux systèmes BSD, ils ne sont souvent connus que des spécialistes. Essayons de dissiper le brouillard qui semble entourer tout cela.

Aux temps préhistoriques

Unix est un système d'exploitation dont la première version date de la fin des années 1960. Il a été mis au point par *Bell Labs*, le centre de recherche de l'opérateur téléphonique historique américain *American Telephone and Telegraph* (AT&T). Très rapidement, Unix a été distribué à l'extérieur d'AT&T sous forme de code source. C'est ce qui a causé l'apparition d'une véritable famille de systèmes Unix, chaque entreprise détentrice d'une licence Unix faisant évoluer le système en ajoutant ses idées et en reprenant celles du voisin.

Une des plus anciennes branches de la famille Unix a été développée à l'Université de Californie à Berkeley (UCB), à partir de 1977. Ces systèmes étaient connus sous le nom de *Berkeley Software Distribution* (BSD). À bien des égards, BSD était la branche de recherche d'Unix. On lui doit l'intégration de nombreuses fonctionnalités, dont par exemple l'implémentation originale de TCP/IP, dans 4.2BSD en 1983. L'Unix BSD a été utilisé comme source de nouveautés ou tout simplement comme point de départ par de nombreux systèmes Unix.

Une autre branche assez répandue est la branche UNIX System V, largement vendue par AT&T à des constructeurs de machines tels que Sun, IBM, HP, ou SGI. Chacun de ces constructeurs a ensuite créé son Unix à partir de cette première version.

Des Unix libres

On compte aujourd'hui plusieurs systèmes Unix libres. Trois d'entre eux descendent de BSD : NetBSD, FreeBSD, et OpenBSD. Le système GNU/Linux est quant à lui une ré-implémentation complète dont l'inspiration provient surtout d'UNIX System V.

Il existe d'autres systèmes Unix libres. Citons Darwin, qui constitue les couches Unix de MacOS X, ou des systèmes plus expérimentaux tels que xMach ou GNU/-Hurd.

CULTURE **Les sources**

Les sources d'un programme sont un ensemble de fichiers contenant du code, que l'on compile pour obtenir le programme exécutable. Unix a toujours eu une culture de distribution sous forme de code source : les entreprises ayant acquis la licence adéquate y avaient accès, ce qui leur permettait de le modifier à leur guise.
La forme binaire est l'autre forme de distribution. Le programme, déjà compilé, est prêt à l'emploi, mais on ne peut pas le modifier.

CULTURE **Les descendants de BSD et de System V**

À la fin des années 1980, la famille Unix commence à s'étoffer. On y trouve les dérivés de System V, comme AIX d'IBM et HPUX de HP. Les dérivés de BSD sont assez nombreux : IRIX de SGI, Digital UNIX de Digital, SunOS de Sun, NeXTStep de NeXT… Certains changent de famille : SunOS et IRIX deviennent par la suite des dérivés de System V, mais en général, tout le monde s'inspire des idées du voisin.

VOCABULAIRE **Linux et GNU/Linux**

Quelle est la différence entre Linux et GNU/-Linux ? Linux est le noyau du système, c'est-à-dire le composant qui attribue les ressources de la machine à tous les programmes, alors que GNU/Linux est le système d'exploitation dans son ensemble, donc le noyau flanqué de nombreux programmes utilitaires.

Quel système est Unix ?

La propriété intellectuelle liée à Unix est passée dans les mains de nombreuses sociétés, dont Novell, qui en 1993 revend le droit de distribuer le code source d'Unix à SCO (*the Santa Cruz Operation*), et la marque déposée Unix au consortium X/Open (lequel sera rebaptisé plus tard *Open Group*).

L'*Open Group* a pour but la standardisation des systèmes Unix. Il publie pour cela des normes dictant les comportements des commandes et des interfaces de programmation. La détention de la marque déposée Unix permet à l'*Open Group* de n'accorder la dénomination « Unix » qu'aux systèmes conformes à ces normes.

Il existe ainsi plusieurs manières de définir Unix, ce qui permet d'alimenter des débats sans fin. Certains sont pour une interprétation stricte de la marque : sont des Unix les seuls systèmes qui ont passé la certification de l'*Open Group* donc présentant la marque Unix. À ce compte, seuls les Unix des constructeurs sont des Unix ; GNU/Linux et les BSD libres n'en sont que des dérivés.

D'autres retiennent l'origine, et accordent le titre d'Unix aux BSD libres, car ils descendent de BSD, qui lui-même descend de l'Unix originel. Mais ils refusent à GNU/Linux l'appellation Unix, puisqu'il a été créé de toutes pièces, sans code issu de l'Unix originel. Ce point de vue amène à différencier Linux et Unix.

Enfin, d'autres vont retenir l'aspect familial, le plus large, et appeler Unix les systèmes ayant un ensemble de comportements et de fonctionnalités en commun. Des systèmes comme GNU/Linux ou les BSD libres font ainsi partie de la famille Unix. C'est ce point de vue que nous retiendrons dans ce livre, afin d'éviter d'avoir à faire d'incessantes distinctions entre « systèmes Unix » et « systèmes dérivés d'Unix ».

Pour résumer, rechercher un véritable système Unix aujourd'hui revient à chercher un véritable Gaulois en France. Tout a progressé, tout s'est mêlé, et pour utiliser Unix, il faut choisir un système Unix particulier. Chaque système contient des éléments issus du tronc commun Unix, et d'autres éléments qui lui sont propres. Dans cet ouvrage, on a choisi de s'intéresser aux systèmes BSD ; nous expliquerons pourquoi dans la section suivante.

CULTURE **SCO et Novell**

À l'heure où ces lignes sont écrites, une guerre de communication fait rage entre SCO et Novell, chacun prétendant avoir des droits sur les sources d'Unix. Cette discorde s'inscrit dans une affaire mettant en jeu SCO contre IBM, pour une obscure affaire de portions du code source d'Unix qui auraient été incluses dans Linux.
Au fil des semaines, la plus grande confusion s'instaure dans cette affaire, et l'auteur ne sait plus très bien quels droits SCO détient réellement sur les sources d'Unix. Les procès à venir et les précisions qu'apportera SCO devraient dissiper les malentendus.

ATTENTION **Un point de vue de plus**

Mentionnons également le cas des nombreux enquêteurs qui vérifient la conformité d'une installation à l'informatique « standard », que tout le monde est censé pratiquer. Ceux-là doivent faire rentrer tout système dans une case : soit « Windows » (sous-entendu sur PC), soit « Unix » (sous-entendu sur station Unix constructeur), soit « Linux » (sous-entendu PC sous GNU/Linux).
La distinction qu'ils marquent entre « Unix » et « Linux » est un point de vue de plus à connaître, même s'il est assez inexact.

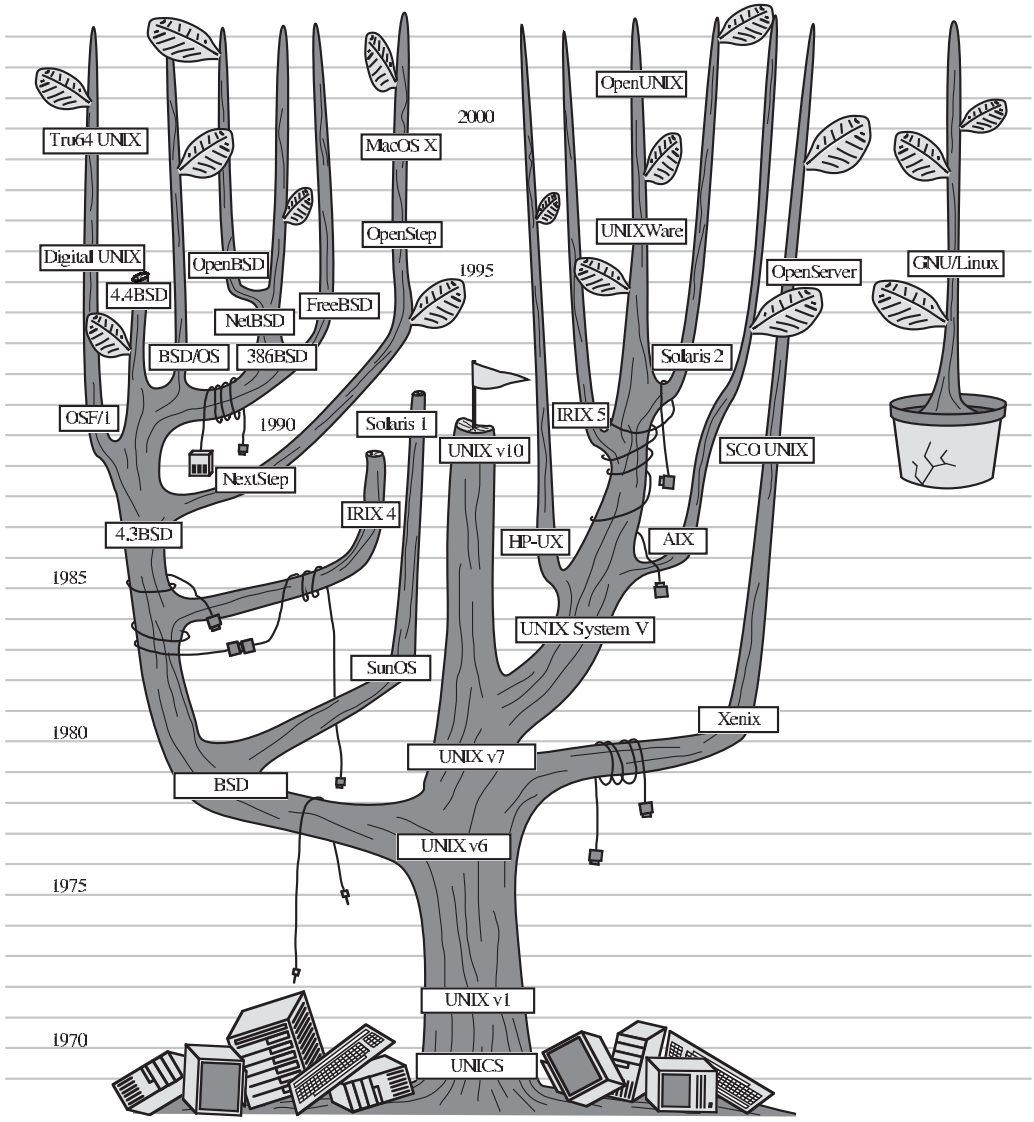
VOCABULAIRE **Unix-like**

En anglais, les systèmes dérivés d'Unix mais ne pouvant pas prétendre à la dénomination Unix *stricto sensu* sont souvent appelés « *Unix-like* », ce qui se traduit assez bien par « système à la Unix ».

B.A.-BA **Logiciel libre**

Le mouvement du logiciel libre a été lancé par Richard Stallman via la *Free Software Foundation* (FSF). Ces logiciels libres fournissent à leurs utilisateurs quatre libertés fondamentales : exécuter le programme, pour tous les usages ; étudier le fonctionnement du programme et l'adapter à ses besoins ; redistribuer des copies du programme ; et améliorer le programme et publier ces améliorations. L'accès au code source du programme est une condition nécessaire.
Un mouvement plus récent, l'*Open Source Initiative* (OSI), propose une définition en dix points qui dans la pratique est quasiment équivalente à celle du logiciel libre selon la FSF. Le contraire de « logiciel libre » est « logiciel propriétaire » : est propriétaire tout logiciel qui n'est pas libre, car il ne remplit pas au moins l'une des quatre libertés fondamentales.
Le projet lancé par Richard Stallman s'appelle GNU (GNU N'est pas Unix). Le but était tout d'abord de concevoir un système d'exploitation libre complet. De nombreux logiciels à ce titre nécessaires furent développés mais le noyau manqua jusqu'à l'apparition de Linux, en 1991. Les logiciels GNU fonctionnent désormais avec d'autres noyaux (Hurd, NetBSD). La FSF s'intéresse désormais aux applicatifs et à la documentation libre.

CULTURE **Un arbre généalogique simplifié d'Unix**



Une représentation en arbre est assez inadéquate pour montrer toutes les influences d'une branche à l'autre : les branches BSD et System V ont régulièrement alimenté les autres Unix, et les récupérations de code entre Unix libres sont monnaie courante.
Les migrations de BSD vers System V sont également représentées de façon peu satisfaisante. IRIX et Solaris n'ont bien entendu pas tout abandonné lors de cette transition. La représentation des branches BSD de ces systèmes comme des branches mortes est donc excessive.
Enfin, il est nécessaire de mentionner le fait que la taille des branches ne représente pas leur importance culturelle ou en nombre d'installations. Les feuilles et les câbles sont purement décoratifs.

Les systèmes BSD

Cet ouvrage aborde des bases d'administration système Unix, en s'appuyant sur les systèmes NetBSD, FreeBSD et OpenBSD. Contrairement à ce que la similitude de leurs noms pourrait laisser croire, ces trois systèmes sont bien distincts, tant au niveau du code source que des fonctionnalités qu'ils proposent.

Ces systèmes ont néanmoins quelques points communs. Un air de famille, pourrait-on dire. Les BSD sont des systèmes Unix : ils sont performants, fiables, et mûrs. Ce sont des systèmes modernes, qui prennent en charge des technologies récentes, comme USB ou IP version 6. Ils sont capables de faire fonctionner des applications telles qu'OpenOffice.org, Mozilla, KDE, ou GNOME, tout comme ils peuvent servir de nombreux protocoles : le Web, FTP, le DNS, les news, etc.

Tous trois sont de plus des systèmes Open Source, libres et gratuits : tous leurs sources et binaires peuvent être téléchargés librement sur l'Internet.

Les trois BSD partagent encore le même mode de développement centralisé, où une unique équipe de développeurs écrit l'ensemble du système et l'intègre en une distribution. Par opposition, les différents éléments des systèmes GNU/Linux sont développés par des équipes nouant peu de liens. Des projets rassemblent ces morceaux et apportent leur propres contributions pour constituer des « distributions Linux ». Le mode de développement des systèmes BSD leur confère ainsi une grande qualité dans l'intégration et la cohérence du système.

En conséquence, et contrairement aux distributions GNU/Linux, les systèmes BSD n'ont chacun qu'une seule distribution : il existe un seul NetBSD, un seul FreeBSD, et un seul OpenBSD. Les distributions GNU/Linux, quant à elles, sont très nombreuses : citons Red Hat, Debian, Slackware, SuSE, etc. On peut y voir un avantage ou un handicap. Quoi qu'il en soit, le dilemme du choix d'une distribution ne se posera pas à ceux qui auront choisi BSD.

Un des avantages principaux des BSD est d'avoir su rester assez proches du tronc commun Unix. Ces systèmes introduisent peu de commandes ou de formats de fichiers particuliers, ce qui rend les compétences acquises sur un système BSD facilement transférables sur un autre Unix, tel que GNU/Linux ou Solaris. J'irai

CULTURE **Histoires de familles**

Il existe d'autres systèmes BSD. Citons Darwin, qui constitue les couches Unix de MacOS X, ou le projet xMach. Ces deux systèmes sont Open Source, mais ils sont assez différents de NetBSD, FreeBSD et OpenBSD. Ils reposent en effet sur le micro-noyau Mach plutôt que sur un noyau Unix monolithique traditionnel.
On trouve également des systèmes Open Source dérivés de FreeBSD et OpenBSD : DragonflyBSD, fondé par une figure historique de FreeBSD pour cause de désaccords sur des choix techniques, et EkkoBSD, démarré à partir des sources d'OpenBSD pour pouvoir travailler « plus démocratiquement ». Ces systèmes bénéficient pour l'instant de très peu de moyens humains, et il est difficile de prédire leur évolution.
Enfin, il existe également un BSD propriétaire : BSD/OS, distribué par la société BSDI, mais cette dernière a annoncé son intention d'en arrêter le développement.

CULTURE **Applications sous Unix**

Si vous êtes totalement étranger au monde Unix, les noms des applications citées ici ne vous diront peut-être rien. Sous Unix, pas de Microsoft Office, pas d'Internet Explorer, pas d'Outlook. Mais l'offre logicielle est vaste (et souvent gratuite). Associons donc quelques fonctions aux noms. OpenOffice.org est une suite bureautique se posant en concurrent de Microsoft Office (il en existe d'ailleurs une version pour Windows). Mozilla est un navigateur Web. KDE et GNOME sont des environnements de bureau évolués et intégrés.

CULTURE **Unix constructeur**

Par ce terme, on désigne les Unix développés par les fabricants de machines. Le tableau ci-dessous donne un aperçu rapide des Unix constructeur, des processeurs sur lesquels ils fonctionnent, et de leurs fabricants.

Fabricant	Système	Processeur
Sun	Solaris	Sparc
IBM	AIX	POWER/PowerPC
SGI	IRIX	MIPS
Digital/Compaq	Tru64 Unix	Alpha
HP	HPUX	PA-RISC/Itanium

même jusqu'à dire, à titre personnel, que j'ai plus appris sur GNU/Linux en utilisant BSD qu'en utilisant GNU/Linux lui-même...

Bien sûr, installer et administrer un BSD ne fera pas de vous un expert Unix tous systèmes confondus. La famille Unix comporte quelques moutons à cinq pattes, qui ne font rien comme les autres, tels qu'AIX d'IBM. Mais en maîtrisant l'administration d'un système BSD, vous ferez l'acquisition d'une culture Unix et de méthodes de travail qui vous permettront de prendre en main relativement facilement la plupart des autres systèmes Unix.

Voyons maintenant les principales caractéristiques de nos trois systèmes BSD.

NetBSD : la portabilité avant tout

NetBSD, fondé en 1993 à partir du projet 386BSD, est l'aîné des systèmes BSD existant actuellement. Il a pour but de créer un système d'exploitation Unix libre, gratuit, stable, performant, sûr, et surtout multi-plateformes.

Au moment de la version 1.0, NetBSD proposait déjà six architectures, utilisant quatre types de processeurs différents : amiga, hp300, i386, mac68k, pc532, et sparc. Une décennie plus tard, NetBSD 1.6 fonctionne sur plus de cinquante plateformes différentes et sur plus de dix types de processeurs.

NetBSD est donc le système de choix lorsqu'il faut redonner un second souffle à une vieille station dont le fabricant a abandonné la prise en charge dans les versions récentes de son système. NetBSD peut transformer un Macintosh II ou une vieille station Sun en un pare-feu ou un petit serveur Web tout à fait valable. C'est aussi le seul BSD capable de fonctionner sur des assistants personnels (PDA) tels que le Jornada de HP, que l'on peut admirer sur la figure 1.1.

Pour autant, il serait réducteur de cantonner NetBSD aux vieilles machines et aux PDA. NetBSD fonctionne très bien sur un certain nombre de serveurs à processeurs d'architecture 64 bits. Il fut par exemple le premier système libre à fonctionner sur stations alpha, ainsi qu'à exploiter ces machines en mode multi-processeurs. Tout aussi à l'aise sur compatibles PC, il peut être utilisé pour mettre en place des serveurs ou des stations de travail à peu de frais.

NetBSD est également très compatible. Avec lui-même tout d'abord : un grand soin est apporté à la compatibilité ascendante. Ainsi, NetBSD 1.6 est encore capable de faire fonctionner des programmes compilés pour NetBSD 0.8. NetBSD cultive aussi la compatibilité avec les autres systèmes : il est capable d'exécuter de nombreux programmes compilés pour d'autres Unix, comme par exemple, FreeBSD, GNU/Linux, Solaris, SCO Unix, Digital UNIX... Ceci lui permet de bénéficier de nombreuses applications qui ne lui étaient pas originellement destinées.

La grande portabilité de NetBSD résulte de l'attention portée à la qualité de son développement : avant tout, les choses y sont bien faites. C'est en adoptant les meilleures solutions – quitte à passer plus de temps à les mettre en œuvre – que l'équipe de NetBSD peut obtenir un système où les mêmes codes sources pilotent

VOCABULAIRE **Portabilité**

La « portabilité » est un anglicisme désignant la capacité d'un logiciel à fonctionner sur différents systèmes d'exploitation ou d'un système d'exploitation à fonctionner sur différentes plateformes matérielles, sous réserve de recompilation ou d'adaptations légères et simples, souvent automatisées.

CULTURE **386BSD**

386BSD est le maillon qui mène du système BSD, développé à Berkeley, aux BSD libres modernes. Ce projet a été abandonné après le lancement de NetBSD et FreeBSD.

ATTENTION **Compatibilité binaire**

La compatibilité binaire ne concerne que des programmes compilés pour le même processeur mais pour un système d'exploitation différent. Ainsi, NetBSD/i386 pourra exécuter les binaires Linux/i386, mais pas les binaires Linux/powerpc – et inversement.

VOCABULAIRE **Pare-feu**

« Pare-feu » est la traduction officielle de l'anglais « *firewall* », qui se traduirait littéralement par « mur antiflammes », terme désignant à l'origine une plaque placée entre le moteur et l'habitacle d'un véhicule afin de réduire la vitesse de propagation d'un incendie. Il existe d'autres traductions moins usitées, telles que « coupe-feu », « barrière de sécurité », ou encore le très élégant « garde-barrière ».

Figure 1–1 NetBSD sur le Jornada 720 de HP. La disquette sert uniquement d'indicateur de l'échelle

SUR LES AUTRES UNIX **NetBSD comparé à Linux**

En matière de portabilité, le seul concurrent sérieux de NetBSD est Linux. Lui aussi a été porté sur un très grand nombre de plateformes, mais ces différentes versions (appelées « ports ») de Linux partagent moins de code que les ports de NetBSD. Par exemple, ils partagent peu les pilotes de matériel, ce que les auteurs de NetBSD s'efforcent de faire. Davantage testé, NetBSD a donc toujours tendance à mieux fonctionner sur les plateformes les moins populaires, comme par exemple mac68k.

plus de 50 architectures différentes. On dispose ainsi d'un code source très bien écrit et agréable à lire, ce qui a son importance quand on souhaite le reprendre.

La compatibilité ascendante est une autre conséquence de cette politique, avec des implications immédiates pour l'administrateur. Ainsi, NetBSD lui épargne autant que possible le désagrément des programmes et des fichiers de configuration à mettre à jour en même temps que le système.

Le lecteur attentif aura peut-être remarqué un net parti pris en faveur de NetBSD dans cet ouvrage. Cela n'a rien de surprenant : l'auteur est membre de l'équipe de développement du projet NetBSD.

FreeBSD : le spécialiste du PC

Le projet FreeBSD a vu le jour peu après NetBSD, dans le but identique de fournir des modifications non officielles à 386BSD. Pourquoi démarrer un projet différent ? Parce que les motivations étaient distinctes. L'équipe de NetBSD voulait un système fonctionnant sur le plus grand nombre de machines possible ;

CULTURE **Des processeurs**

Outre les 80x86 d'Intel, qui équipent les PC classiques, et les 680x0 de Motorola, qui ont équipé de nombreuses machines, dont les anciens Macintosh d'Apple, le monde des processeurs regorge de puces créées par les fabricants de machines Unix. Citons le Sparc pour les stations Sun, le MIPS pour les stations SGI, le POWER et le PowerPC pour les stations IBM (également utilisés dans les nouveaux Macintosh), le PA-RISC et l'Itanium pour les stations HP, et l'Alpha et le Vax pour les stations Digital.
L'informatique embarquée a elle aussi apporté de nouvelles puces, comme le SuperH ou les ARM.

celle de FreeBSD préférait un système fonctionnant au mieux sur une plateforme donnée : le compatible PC.

FreeBSD s'est donc concentré sur l'optimisation pour les PC. Ainsi, il a été capable d'utiliser des PC multi-processeurs plusieurs années avant NetBSD. FreeBSD reconnaît également un grand nombre de cartes d'extension pour PC, beaucoup plus que NetBSD.

FreeBSD est aussi plus connu, ce qui lui confère davantage d'utilisateurs, et donc plus de bras pour réaliser des paquetages. Le système de paquetages de FreeBSD est par conséquent le plus riche des trois systèmes BSD. N'en concluez pas pour autant que les autres sont pauvres ou inutilisables !

Enfin, FreeBSD a une véritable avance sur les autres BSD en ce qui concerne la compatibilité binaire avec Linux. Il peut accueillir des programmes tels que Matlab ou Oracle pour GNU/Linux, alors que les autres BSD échouent faute de compatibilité binaire complète.

Finissons par un mot sur l'aspect multi-plateformes : FreeBSD s'est un peu diversifié. Il dispose d'une version pour machines Alpha bien rôdée, et des versions pour Sparc, PowerPC et Itanium, plus récentes – donc moins éprouvées.

CULTURE **Oracle et Matlab**

Oracle est un puissant logiciel de base de données, proposant des fonctionnalités très complexes comme la répartition d'une base de données sur plusieurs machines, pour gagner en fiabilité. Matlab est un environnement de calcul scientifique et technique précieux dans certains domaines. Ces deux logiciels sont propriétaires.

ATTENTION **Tout évolue rapidement**

C'est une caractéristique des systèmes libres : la situation peut changer très vite. Même si l'on peut supposer qu'au moment où vous lirez ces lignes, FreeBSD comptera toujours plus de paquetages que NetBSD et OpenBSD, il est possible qu'entre-temps quelqu'un ait décidé de travailler sur la compatibilité binaire de NetBSD ou OpenBSD, et que ces systèmes soient donc devenus capables d'exploiter Oracle ou Matlab compilés pour GNU/Linux.

OpenBSD : l'obsession de la sécurité

NetBSD et FreeBSD sont des projets distincts pour des raisons d'orientation technique, mais OpenBSD est né de différends d'ordre personnel. Le fondateur d'OpenBSD était un membre de l'équipe dirigeante de NetBSD et en a été évincé par ses pairs. En 1995, il a donc démarré son propre BSD : OpenBSD, un système reposant sur NetBSD 1.0.

Les partisans de chaque camp ont leurs versions des faits quant aux motifs de cette séparation. Pour bénéficier d'un point de vue neutre sur la question, le plus simple est de prendre son courage à deux mains et de lire les centaines de courriers électroniques des archives des listes de diffusion où les coups se sont échangés.

OpenBSD dérivant de NetBSD, il en est très proche. La séparation n'a pas eu lieu pour des motifs techniques, et ce fut la sécurité qui lui permit de se démarquer.

Au moment de sa sortie, OpenBSD bénéficiait d'un avantage certain sur ses deux cousins BSD : NetBSD et FreeBSD étaient basés aux États-Unis d'Amérique et devaient se conformer à des lois strictes, interdisant l'exportation d'algorithmes de cryptographie forte, alors assimilés à des armes de guerre. Installé au Canada, OpenBSD pouvait quant à lui distribuer librement tous les outils de cryptographie qu'il souhaitait. Cette situation a duré plusieurs années, conférant à OpenBSD un avantage indéniable sur le plan de la sécurité. Mais les lois américaines ont depuis été assouplies, et les BSD sont désormais tous trois distribués avec une panoplie complète d'outils de cryptographie forte.

OpenBSD a fait beaucoup d'efforts sur le plan de la sécurité, en particulier en conduisant un audit du code hérité de NetBSD, et en en corrigeant de très nombreuses failles. Mais il ne faut pas imaginer que les autres BSD se sont contentés

de rester passifs : dans chaque projet BSD, on trouve des développeurs qui surveillent de près les modifications des voisins, et qui réintègrent les corrections de bogues qu'ils observent. La recopie est de mise, c'est ainsi que le logiciel libre fonctionne.

Lors de la première édition de ce cahier, OpenBSD avait par exemple pris une avance significative sur NetBSD et FreeBSD pour réduire l'impact de certains trous de sécurité, en imposant une pile non exécutable sur les architectures qui le permettent. Les deux autres BSD ont depuis travaillé à l'intégration de cette fonctionnalité : les innovations sécuritaires sont trop intéressantes pour que le système où elles ont été mises au point puisse seul s'en prévaloir très longtemps.

On doit aussi au projet OpenBSD le développement d'outils de sécurité tels qu'OpenSSH, actuellement très utilisé même au-delà de la sphère BSD.

Tous ces efforts louables ont contribué à améliorer la sécurité des trois BSD et d'autres systèmes Unix, mais ils n'ont pas forcément fait d'OpenBSD un système plus sûr que les autres BSD. Le projet OpenBSD clame haut et fort sur sa page d'accueil n'avoir eu qu'un seul trou de sécurité depuis 1997 dans son installation par défaut. C'est une escroquerie intellectuelle : personne n'utilise un système dans sa configuration par défaut. On utilise une machine Unix pour employer des programmes clients et serveurs, et quand on les démarre, on quitte la configuration par défaut, exposant ainsi de nouveaux trous de sécurité. OpenBSD n'en a pas forcément moins que FreeBSD ou NetBSD, et pour cause : les applications que l'on emploie sur ces systèmes sont souvent les mêmes.

De plus, beaucoup de trous de sécurité sont ouverts par l'administrateur lui-même, quand il fait des erreurs de configuration et ouvre grand la porte aux pirates. Il ne faut donc pas se leurrer : même si le projet OpenBSD fait un travail remarquable sur le plan de la sécurité, ne croyez pas que ce système est intrinsèquement sûr. Comme tout programme écrit par des humains, il contient des erreurs, dont certaines ont des implications sur la sécurité. Utiliser OpenBSD ne vous dispensera donc pas de tenir à jour votre système, et comme les développeurs d'OpenBSD sont très forts pour trouver et corriger des bogues, les mises à jour seront nombreuses.

Pour ce qui ne relève pas de la sécurité, OpenBSD est un projet de taille nettement inférieure à FreeBSD et NetBSD. Le faible nombre de développeurs a pour conséquence un système de paquetages moins fourni que celui de NetBSD et FreeBSD – il n'est pas pour autant ridicule. L'utilisation d'OpenBSD comme machine de bureautique est toutefois plus difficile à mettre en œuvre qu'avec NetBSD et FreeBSD, à cause du faible nombre de paquetages disponibles pour cette utilisation.

Signalons un autre point important : OpenBSD assure très peu de compatibilité ascendante. À chaque nouvelle mise à jour, il faut suivre rapidement. Certains développeurs du projet OpenBSD prétendent que ces ruptures de compatibilité ont un but sécuritaire : forcer les gens à faire les mises à jour les contraint à récupérer les corrections des trous de sécurité dont souffrait la version précédente. On peut aussi imaginer que le faible nombre de développeurs n'aide pas à maintenir plusieurs versions du système d'exploitation à la fois.

PLUS LOIN **La pile et les débordements de variables**

Un trou de sécurité très classique consiste à oublier de vérifier les bornes d'une variable dans un programme. Un attaquant peut exploiter cet oubli en allant écrire à certains endroits non prévus de la zone de mémoire où sont stockées les variables locales (cette zone s'appelle la pile).
L'attaquant dépose du code sur la pile, et il ne lui reste plus qu'à trouver un moyen de le faire exécuter pour détourner le programme. En imposant une pile non exécutable, on minimise les conséquences d'un mauvais contrôle de bornes, puisque le code écrit sur la pile ne pourra pas être exécuté. Cette mesure est extrêmement efficace contre toute une classe d'attaques, mais elle ne peut être mise en œuvre qu'avec les processeurs capables de rendre une zone de mémoire accessible en lecture/écriture mais pas en exécution.

CULTURE **OpenSSH**

OpenSSH est une implémentation libre de SSH, un protocole permettant d'assurer des connexions sécurisées entre deux machines, pour faire par exemple de l'administration système à travers Internet.

CULTURE **Compatibilité ascendante**

Les logiciels, architectures, et systèmes d'exploitation évoluent au cours du temps. Quand plusieurs éléments fonctionnent de concert, ils doivent être compatibles. On appelle compatibilité ascendante le fait qu'un nouveau produit, mettant en place une nouvelle norme, continue à comprendre et interagir avec une norme plus ancienne, qu'il rend pourtant obsolète. Ainsi, du code compilé pour processeur i386 ou i486 fonctionnera sur des Pentium ; un lecteur de DVD comprendra les CD-ROM ; etc.
L'inconvénient de la compatibilité ascendante est évidemment le volume et la complexité sans cesse croissants de l'héritage à prendre en compte. Certains projets logiciels, plus soucieux de simplifier les choses que de ménager leurs utilisateurs, font régulièrement table rase du passé, imposant ainsi une mise à jour et une reconfiguration plus complètes.

Lequel est le meilleur ?

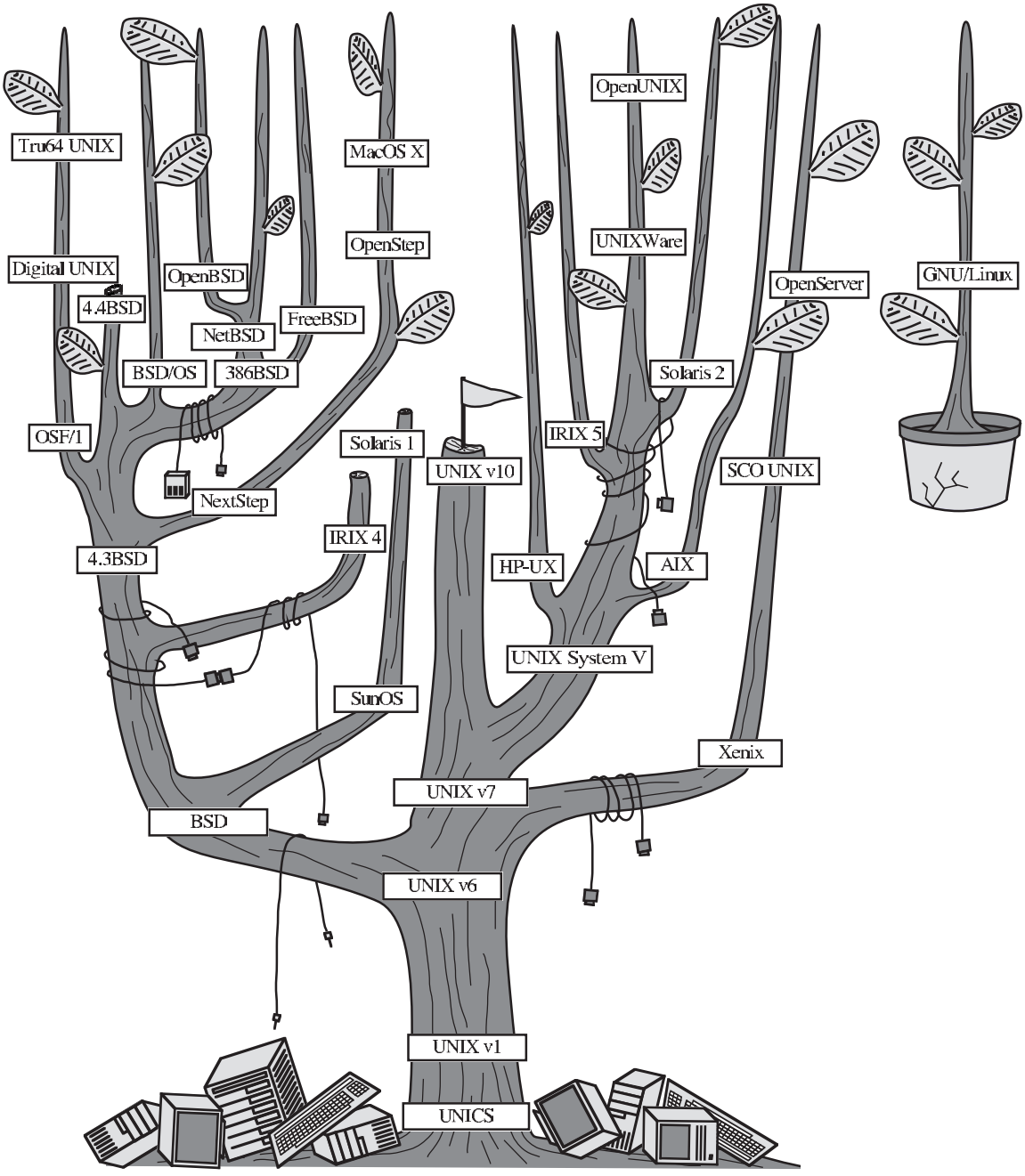
C'est un fait avéré : le public a horreur de la diversité. Puisqu'il y a trois BSD, il me faut obligatoirement indiquer lequel est le meilleur, ce qui permettra de négliger les deux autres.

La vie n'est pas si simple, et chacun des BSD a ses avantages. C'est ce qui justifie leurs existences même ! Certaines personnes, qui trouvent rassurant de ranger les choses dans des petites boîtes, vous expliqueront peut-être que suivant l'utilisation que l'on en fait, un seul BSD est valable. On entend par exemple souvent que du matériel exotique impose le choix de NetBSD, que pour l'utilisation sur PC il faut retenir FreeBSD, et que seul OpenBSD mettra en place un pare-feu convenable.

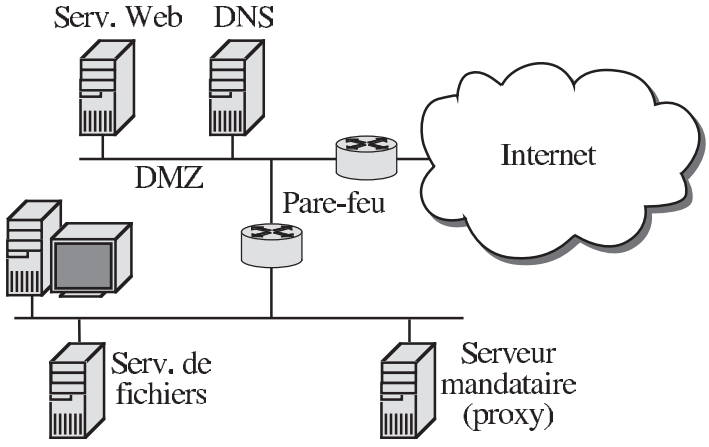
Ce genre de classement ne rime pas à grand-chose. OpenBSD reconnaît du matériel exotique, NetBSD et OpenBSD fonctionnent très bien sur PC, et ces trois systèmes font de très bons pare-feu. Mais tout cela ne vous dit pas lequel choisir...

L'idéal serait de tester les trois. À défaut, le conseil de l'auteur va vers NetBSD, mais n'oubliez pas que son jugement est biaisé. NetBSD vous procurera un environnement performant et fiable quels que soient la mission et le matériel que vous lui confierez. En particulier, vous pourrez l'employer tout à la fois sur une vieille machine sauvée du rebut pour vous faire la main, sur un PDA, ou sur un gros serveur.

Autre avantage, les exemples de ce cahier ont plus tendance à s'appuyer sur NetBSD que sur ses deux cousins, autre conséquence de la préférence de l'auteur...


Tru64 UNIX
Digital UNIX
4.4BSD
OSF/1
OpenBSD
NetBSD
BSD/OS
386BSD
FreeBSD
MacOS X
OpenStep
NextStep
4.3BSD
IRIX 4
Solaris 1
SunOS
BSD
UNIX v10
HP-UX
UNIX System V
UNIX v7
UNIX v6
UNIX v1
UNICS
OpenUNIX
UNIXWare
IRIX 5
Solaris 2
AIX
OpenServer
SCO UNIX
Xenix
GNU/Linux

2


Serv. Web
DNS
Internet
DMZ
Pare-feu
Serv. de
fichiers
Serveur
mandataire
(proxy)

Présentation de l'étude de cas

Plusieurs raisons peuvent vous amener à vous intéresser à l'apprentissage des bases d'Unix avec un système BSD. Examinons-en quelques-unes à l'aide d'une étude de cas.

SOMMAIRE

Unix domestique

Première situation, vous désirez découvrir Unix sur une configuration montée à la maison, probablement par curiosité personnelle. Vos motivations peuvent être assez diverses :

- Vous êtes frustré par l'informatique microsoftienne, coûteuse et peu fiable. On vous a présenté Linux comme le remède miracle, ce qui a piqué votre curiosité : quel dommage ce serait de manquer une telle panacée !
- Vous avez longtemps utilisé les services Internet (le Web, IRC, les news, la messagerie...) comme client, et avez voulu tester les choses côté serveur. Après avoir expérimenté quelques serveurs sur votre système d'exploitation habituel, vous avez la sensation de passer à côté de quelque chose. Vous voulez donc expérimenter tout cela sous Unix, par acquit de conscience.
- Vous savez que les compétences Unix se vendent bien sur le marché du travail, et avez donc décidé de l'inclure sur votre CV. Par souci d'honnêteté, vous allez d'abord l'apprendre.

Quelle que soit votre motivation, il est probable que vous ayez d'abord tenté votre chance avec GNU/Linux : il est gratuit et célèbre. Si tel n'est pas le cas, c'est presque dommage, car vous risquez de ne pas remarquer tous les avantages de BSD. Mais ce qui suit devrait quand même vous aider un peu à voir ce que vous avez raté.

Vous avez donc installé votre machine en double amorçage avec GNU/Linux, puis commencé à faire quelques allers-retours avec votre système habituel pour consulter la documentation en ligne qui vous permettrait de configurer l'accès à Internet. À chaque nouvel échec, un redémarrage s'imposait.

B.A.-BA **Double amorçage**

Le double amorçage est une configuration où la machine est capable de démarrer sur deux systèmes d'exploitation différents. En général, on met en place un menu de démarrage qui donne le choix du système à invoquer. En anglais, on parle de *dual-boot*.

Une fois votre installation de GNU/Linux à peu près fonctionnelle, vous avez constaté que c'était très joli, très coloré, avec une belle interface graphique, mais que votre maîtrise de ce système n'était pas bien supérieure à celle que vous aviez de Windows. Dès votre première utilisation, le système proposait déjà de nombreux services et vous n'aviez pas la moindre idée des relations liant ses divers éléments. Il est même possible qu'il se mettait à jour seul, modifiant son comportement et son organisation à mesure que vous tentiez de le comprendre.

Le système, livré avec de nombreux frontaux d'administration en mode graphique, vous a permis de configurer facilement la couleur du fond de l'écran, mais pas encouragé à essayer de comprendre comment les choses fonctionnaient : pourquoi se compliquer la vie à visiter des arborescences de fichiers de configuration alors que l'on l'impression de tout pouvoir faire à la souris ?

Un jour, quelque chose a mal tourné, et comme vous ne compreniez rien au fonctionnement de tout ce fatras, vous avez appliqué la méthode traditionnelle pour régler un problème sous Windows : tout réinstaller.

Le pire, c'est que vous avez peut-être trouvé tout cela normal. Cela ne l'est pas : sur les systèmes Unix, on peut comprendre comment fonctionne le système, et on peut corriger les problèmes sans tout réinstaller à chaque fois. Encore faut-il,

pour cela, que le système vous aide un peu. C'est là que les BSD peuvent vous apporter quelque chose.

Les BSD vous rendront les mêmes services que GNU/Linux, mais après configuration. Au départ, le système ne fait rien ; aucun service n'est démarré. Tout ce que vous ferez fonctionner sur la machine, vous l'aurez mis en place ; vous en comprendrez donc le fonctionnement et saurez y intervenir. En prime, les connaissances que vous aurez acquises s'appliqueront très bien aux autres Unix, en particulier à GNU/Linux.

PLUS LOIN **Mauvais, GNU/Linux ?**

Il faut tout de même dissiper certains malentendus : GNU/Linux n'est pas un mauvais système. On ne fera pas ici le procès de sa performance ou de sa stabilité, elles sont toutes les deux excellentes.
Ce que l'on peut reprocher aux distributions Linux les plus axées grand public (que vous essaierez si vous en choisissez une au hasard, puisque ce sont les plus répandues), c'est leur volonté de masquer la complexité du système. Pour l'usager qui dispose d'un gourou pour administrer sa machine ou pour l'administrateur confirmé, cela ne pose pas de problème, au contraire.
En revanche, pour l'administrateur débutant, ces choix sont lourds de conséquences : il est très difficile de prendre le système en main dans ces conditions.

Le plan de vol

Utiliser BSD n'est pas la solution à tous vos maux : il y a quelques autres écueils à éviter. Le premier, c'est sans doute le double amorçage, et ses allers-retours entre les systèmes pour pouvoir configurer la connexion à Internet. Vous gagnerez un temps précieux en ayant une deuxième machine, dédiée à BSD. En plus, vous pourrez essayer les services que vous monterez sur la machine BSD depuis votre machine habituelle, ce qui sera très gratifiant.

Seulement voilà : vous n'avez pas les moyens de vous offrir une deuxième machine. Pas de problème : pour apprendre, une vieillerie sauvée des ordures fera tout à fait l'affaire (l'auteur a longtemps fait fonctionner un serveur Web personnel sur une Sun IPC trouvée dans une benne de l'université). Les vieux Macintosh ou PC achetés d'occasion sont eux aussi de bonnes pistes : ils sont lents, mais sous réserve de gonfler un peu leurs configurations mémoire et disque, ils feront des petits serveurs tout à fait valables pour apprendre.

PERFORMANCES **Matériel préhistorique**

Si vous devez acheter une vieille machine d'occasion pour essayer un Unix, pensez tout de même à vérifier qu'elle est capable de l'accueillir. Pour les PC, il vous faudra au minimum un 386 ; pour les Macintosh, un 68030. Il est à noter que les vieux Macintosh sont dotés de disques SCSI, ce qui les rend nettement plus performants que les PC du même âge, bridés par des contrôleurs IDE de première génération.
Il faut aussi avoir à l'esprit que les vieilles machines ne sont pas forcément aussi fantastiques qu'elles peuvent en avoir l'air. S'essayer à Unix sur une machine multipliant les problèmes matériels n'est pas très agréable. De plus, si un Macintosh II ou un 486 sont assez puissants pour jouer le rôle d'un petit serveur, ils risquent d'être assez décevants si vous essayez d'y installer une interface graphique.

Deuxième problème, il vous faut un but. Si vous installez un système Unix pour le plaisir, vous aurez vite fait le tour de la question. Ayez des projets : installer un pare-feu pour votre connexion Internet (chapitres 9 et 10), monter un serveur Web personnel (chapitre 12), ou encore une installation bureautique en logiciel libre (les pistes sont données au chapitre 11).

Avant d'atteindre votre but, il vous faudra acquérir des fondamentaux, valables pour tous les systèmes Unix. Le chapitre 3 vous apprendra l'utilisation de la ligne de commande. Sa maîtrise vous permettra de n'être jamais pris au dépourvu, même quand l'interface graphique refusera de démarrer.

Les chapitres 5 à 8 seront l'occasion d'apprendre un peu d'administration Unix pour maîtriser le fonctionnement du système : comment se passe le démarrage, comment régler les problèmes qui peuvent le gêner, et comment gérer les utilisateurs et les services fonctionnant sur la machine. La configuration du réseau et de l'interface graphique sont abordées dans le chapitre 7.

L'administrateur de systèmes Microsoft

Autre profil, autres besoins. Vous gérez un parc Microsoft, avec des clients Microsoft, ce qui est hélas courant, mais disposez de plus – ce qui est plus dur – de serveurs tous équipés en Microsoft : le serveur Web IIS, la messagerie Exchange,

le serveur DNS de Microsoft, et ainsi de suite. Vous avez hérité cette situation de votre prédécesseur, ou avez monté le réseau en utilisant ce que vous aviez l'habitude d'utiliser pour faire vos rapports et vos présentations : du Microsoft.

Êtes-vous content de votre installation ? Peut-être pas. Tous les nouveaux virus et vers sont pour vous. Certes, les mises à jour vous protégeront, mais il y en a une quantité phénoménale et vous n'avez pas le temps de suivre. Pour couronner le tout, à chaque mise à jour, vous risquez de compromettre le bon fonctionnement de tout l'édifice, somme toute assez capricieux.

Si vous avez beaucoup de serveurs, vous savez combien le déploiement de quoi que ce soit sur toutes les machines est pénible. Vous êtes las de devoir rechercher dans la base de connaissances Microsoft quelle clef de base de registres créer pour modifier tel ou tel comportement. Les messages d'erreur cryptiques vous fatiguent. De plus, tout cela coûte très cher. Vous avez envie d'autre chose.

Vous avez peut-être discuté avec un collègue qui gère des serveurs Unix, et visiblement sa vie est bien moins insupportable. Mais il y a une grosse barrière à l'entrée : il faut connaître Unix. Vous n'êtes pas moins compétent qu'un autre, vous décidez d'apprendre, et de confier à Unix quelques services, s'il fait l'affaire.

Les systèmes BSD peuvent se révéler un bon choix ; ils font d'excellents serveurs. Ils sont gratuits, ce qui vous permettra enfin de faire des économies (si vous avez un budget à dépenser, vous vous rattraperez sur le matériel). Ils sont fiables, performants, bref, tout ce qu'il vous faut. Les mises à jour de sécurité des systèmes BSD tombent à un rythme supportable, et ces derniers bénéficient de l'avantage de la minorité : les vers et les virus sont écrits pour Windows et Linux, mais beaucoup plus rarement pour les BSD.

Même si l'activité professionnelle de votre entreprise vous empêche de choisir le système (par exemple, on a décidé pour vous le couple Solaris/Oracle), un système BSD sera probablement un bon terrain pour faire votre apprentissage.

Le plan de vol

Entre deux coups de fil d'utilisateurs qui n'arrivent pas à imprimer, vous allez donc monter un serveur expérimental. Vous projetez probablement de monter sous Unix un serveur de fichiers ou d'impression, un contrôleur de domaine NT, un serveur DNS, un serveur DHCP, un serveur mandataire, un pare-feu, ou un serveur Web. Rappelez-vous, tout cela ne vous coûtera pas un centime en licences.

Vous allez donc suivre le même parcours que pour Unix à la maison, mais en sautant probablement le chapitre 9, traitant de choses que vous devez déjà connaître : les réseaux Ethernet et TCP/IP. Le chapitre 13 retiendra toute votre attention, puisqu'il est consacré à l'étude des catastrophes informatiques, sujet que vous connaissez bien – votre métier c'est de les éviter.

ATTENTION **Mises à jour**

Le fait que les vers ne soient pas prévus pour votre système ne dispense pas de faire des mises à jour. Cela laisse juste un peu plus de temps avant d'être piraté.

CULTURE **SunOS et Solaris**

Le système d'exploitation développé par Sun s'appelait à l'origine SunOS et c'était un BSD. Sun l'a plus tard rebaptisé Solaris, puis l'a fait virer de bord pour le baser sur un UNIX System V. Ainsi, Solaris 1 est un BSD, et Solaris 2 est un System V.
Par un caprice de numérotation, Sun s'est mis à faire tomber le numéro majeur de version : ainsi, le successeur de Solaris 2.6 s'appelle Solaris 7 ; suivent Solaris 8 et 9.
Pour compliquer encore l'affaire, Sun a continué à donner un nom SunOS à ses systèmes baptisés Solaris. Pour chaque version de Solaris, il y a un deuxième numéro de version qui correspond à SunOS : Solaris 1 correspond à SunOS 4.1.1, Solaris 2.x à SunOS 5.x... La numérotation des versions du système de Sun est extrêmement compliquée.

VOCABULAIRE **Serveur mandataire, ou serveur proxy**

On trouve souvent le terme « serveur *proxy* », qui se traduit officiellement en français par « serveur mandataire ».

CULTURE **Serveurs Unix sous Windows**

Certains serveurs du monde Unix, tel le serveur DNS, sont capables de fonctionner sous Windows. Comme dans la version Unix, ces serveurs ne coûtent rien en licences. Si cette possibilité vous intéresse, il faut savoir que ces logiciels sont peu conformes aux standards d'interface utilisateur de Windows. Une culture Unix aide à les mettre en œuvre.

L'administrateur Unix

Dernier lecteur potentiel, l'administrateur de systèmes Unix constructeur dont le parc vieillit. Vous avez la charge de moyens informatiques d'une entreprise ou d'une université, comprenant de vieilles stations Sun, Silicon Graphics, des Alpha, des VAX, etc.

Vous êtes relativement content de ces machines, elles vous ont rendu des fiers services par le passé, mais l'heure de leur retraite approche. Non pas que le matériel fatigue (il est en pleine forme), mais parce que ces vieux systèmes multiplient les trous de sécurité. Vous n'avez pas envie de payer le prix fort pour une mise à jour du système d'exploitation qui, si elle existe, sera peut-être inadaptée à votre machine, un peu trop vieille.

Vous êtes face à un double problème : d'une part migrer les services vers des machines plus récentes, et d'autre part conserver en état de fonctionnement des applications développées exprès pour vos stations.

Pour vos nouvelles machines, un Unix libre vous assurera un système fonctionnel et pérenne. La migration vers les Unix libres est dans l'air du temps : c'est plus économique. Vous êtes habitué aux outils Unix, et BSD vous plaira sans doute plus que les distributions Linux, car ces dernières semblent jouer à celle qui introduira le plus de changements gratuits dans l'administration du système.

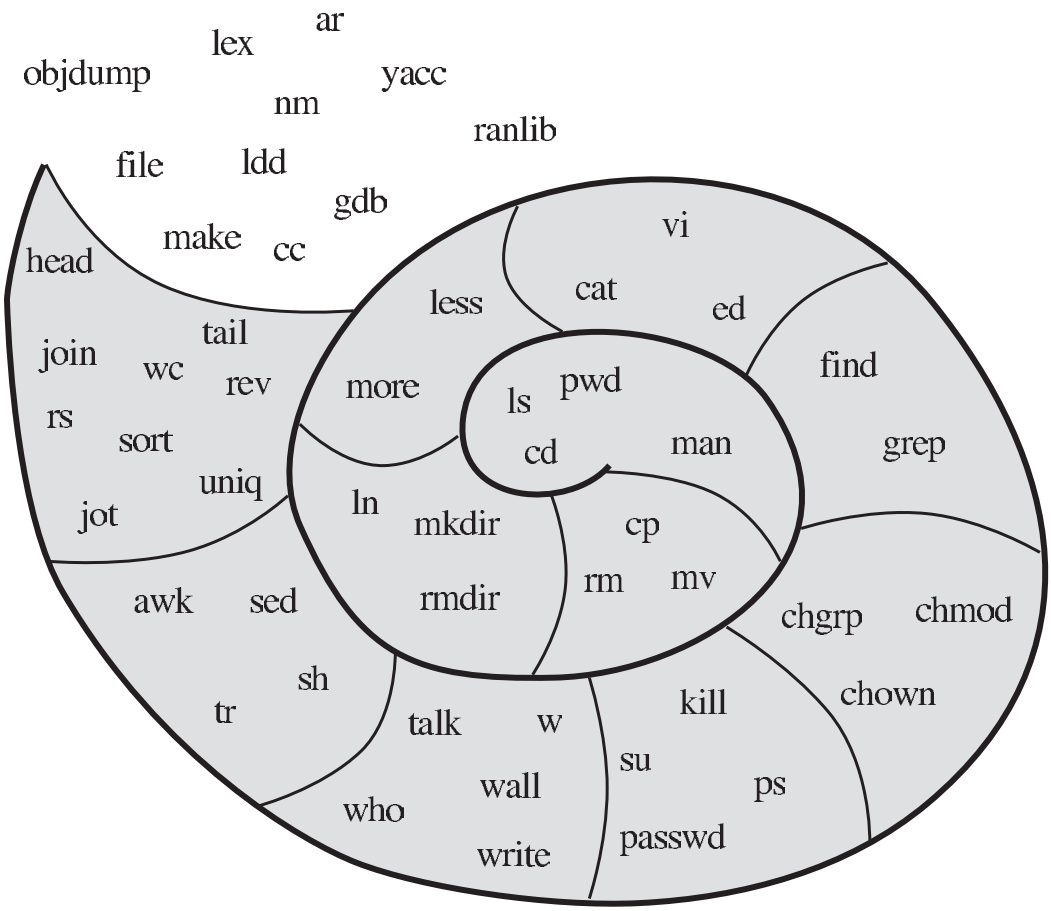
Si vous décidez de migrer vos vieilles stations sous BSD, il est probable que vous pourrez toujours faire fonctionner les programmes compilés pour votre Unix propriétaire, en employant les capacités de compatibilité binaire présentes dans les systèmes BSD. La compatibilité binaire permet de faire fonctionner les applications écrites pour le même processeur mais pour un Unix différent. La compatibilité n'est jamais exacte à 100 %, mais elle permet souvent de faire fonctionner les applications pour lesquelles il n'existe pas d'équivalent en logiciel libre – en fait c'est même sa raison d'être.

Le plan de vol

Vous allez donc monter une machine sous BSD pour utiliser les services anciennement assurés par vos stations de travail. Vous êtes déjà un usager confirmé d'Unix, et vous n'avez pas vraiment besoin de ce livre pour vous débrouiller. Vous voudrez peut-être examiner la séquence de démarrage (chapitre 5) ou la gestion des utilisateurs et des services (chapitres 6 et 8) ; vous y retrouverez des concepts familiers.

Les seuls chapitres qui vous apporteront quelque chose de nouveau sont sans doute celui traitant du système de paquetages et de la compatibilité binaire (chapitre 11) et celui portant sur les mises à jour du système (chapitre 13). C'est peu, mais il en faut bien pour tout le monde !

3


ar
lex
objdump
yacc
nm
ranlib
file
ldd
gdb
make
cc
head
vi
less
cat
ed
tail
join
wc
rev
more
pwd
find
ls
rs
sort
cd
man
grep
uniq
jot
ln
mkdir
cp
rm
mv
awk
sed
rmdir
chgrp
chmod
sh
tr
talk
w
kill
chown
su
wall
ps
who
passwd
write

Bien utiliser Unix pour mieux l'administrer

Nous allons introduire l'utilisation d'un système Unix avec les commandes de base, permettant de manipuler et d'éditer des fichiers, et de gérer les processus en cours. Les usagers déjà familiers de la ligne de commande Unix peuvent sans doute passer directement au chapitre suivant.

SOMMAIRE

MOTS-CLEFS

- Ligne de commande
- Le shell Unix
- Manipulation des fichiers
- Processus

Pourquoi maîtriser la ligne de commande ?

À l'ère des interfaces graphiques prétendument conviviales et ergonomiques, la ligne de commande peut sembler dépassée. Datant des années 1970, elle semble cryptique et complexe au profane. Qu'apporte-t-elle donc par rapport aux interfaces homme-machine se pilotant à la souris ? Cette question a plusieurs réponses.

Expressivité de l'interface texte

Tout d'abord, la ligne de commande est bien plus expressive que n'importe quelle interface graphique. Toute interface utilisateur, graphique ou textuelle, vise à communiquer avec la machine. Une interface graphique limite la complexité des instructions qu'elle permet de construire. Par exemple, à moins d'écrire un programme *ad hoc*, il y est impossible de demander à la machine de « tuer les processus de l'utilisateur toto occupant plus de 4 Mo ».

La ligne de commande bien maîtrisée permet des constructions bien plus complexes. Donnons un exemple percutant.

Pour un fanatique de l'interface graphique, l'équivalent de la ligne de commande suivante sera long à reproduire : j'envoie un courrier électronique à tous les utilisateurs occupant plus de 10 Mo sur leur compte (message préparé dans le fichier homefull.txt)

```
while read pwent ; do user=`echo $pwent|awk '{print $1}'`; home=`echo $pwent |
awk '{print $5}'`; test `du -ks $home|awk '{print $1}'` -ge 10240 &&
mail $user < homefull.txt ; done < /etc/passwd
```

À défaut d'être agréable à lire, cette ligne de commande s'exécute très rapidement : quelques secondes pour une centaine d'utilisateurs. Réaliser la même chose avec une interface graphique sur un système Windows prendrait plusieurs heures, et mettrait à rude épreuve les nerfs de l'administrateur système.

L'économie des ressources

Deuxième argument pour la ligne de commande : sa faible consommation de ressources. Gérer l'interface graphique consomme du temps de calcul et de la mémoire, ressources mieux employées à assurer les services pour lesquels la machine a été mise en place. Pour un serveur où le seul utilisateur de l'interface graphique est un administrateur qui opère en maintenance, le gâchis de ressources est patent.

L'utilisation de l'interface graphique est parfois difficile, voire impossible. Lors de l'administration à distance d'une machine située à l'autre bout du monde, la latence des communications peut dépasser la seconde.

Autre exemple de situation à latence : l'administration d'une machine à travers un lien saturé. La saturation limite la bande passante disponible et augmente la

CULTURE **Communications par satellite**

Lorsqu'elle transite par satellite, l'information se propage à la vitesse de la lumière (environ 300.000 km/s). Or, les satellites sont placés en orbite géostationnaire, à 36.000 km du sol. Pour se propager du sol au satellite puis du satellite au sol, l'information franchit donc 72.000 km ; et cette seule étape dure au minimum un quart de seconde. Les liens satellites peuvent être multiples, d'où des latences encore plus importantes.

latence. Il est fréquent d'observer des délais de plusieurs secondes entre l'action effectuée et la réaction de la machine.

Dans toutes ces situations, une interface graphique se révèle inutilisable. Tout ceux qui ont déjà travaillé sur une machine surchargée et dont l'interface graphique est peu réactive comprendront aisément. Rien n'est plus énervant que d'attendre plusieurs secondes à chaque clic de souris.

L'interface texte souffre elle aussi des augmentations de latence, mais il est toujours possible de travailler sans piquer de crise de nerf. On peut par exemple taper ses commandes en aveugle et attendre patiemment le résultat, taper ses commandes sur la machine locale et les copier/coller dans la session distante, ou préparer des scripts et les envoyer d'un bloc pour exécution sur la machine distante.

CULTURE **Les seaux percés**

Les machines communiquent entre elles par l'envoi de petits « paquets » d'information, appelés datagrammes. L'Internet est composé de réseaux reliés par des routeurs. On modélise parfois les routeurs comme des seaux percés : les paquets entrant dans le routeur sont l'eau versée, et les paquets sortants, l'eau qui s'écoule.
Si un routeur reçoit plus de paquets qu'il ne peut en envoyer sur un lien donné, il mets les paquets en file d'attente : le seau se remplit. Les paquets tardent plus à traverser le routeur, et la latence augmente.
Quand la file d'attente est pleine, le routeur doit abandonner des paquets (le seau déborde). Les paquets perdus devront être retransmis, ce qui augmente encore la latence.

Quand l'interface graphique ne répond plus

Dernier argument en faveur de la ligne de commande : c'est parfois la seule possibilité. Les systèmes BSD, lors de leur installation, ne configurent pas d'interface graphique : il faut pour cela jouer de l'interface texte. De plus, même après configuration d'une interface graphique, le système de base ne contient aucun programme permettant d'administrer la machine via des frontaux graphiques. Il faudra donc installer de nombreux paquetages pour pouvoir se passer éventuellement de l'interface texte.

L'utilisateur ayant découvert l'informatique récemment pourrait s'interroger sur l'intérêt d'utiliser un système BSD, alors que tant de distributions Linux démarrent immédiatement l'interface graphique.

Mais cet usager moderne oublie une chose : tôt ou tard, quand on administre une machine, on finit par casser quelque chose. Ou bien quelque chose tombe en panne : les disques dur meurent, eux aussi. Les coupures de courant peuvent encore amener leur lot de problèmes. Et un jour, la machine, très malade, ne sera plus capable de démarrer l'interface graphique, ni la plupart des services pour lesquels on l'a configurée.

B.A.-BA **Les distributions Linux**

Les distributions Linux abondent, et se divisent en deux catégories : d'une part, celles, faciles à installer, qui finiront par vous faire tourner en bourrique à tout automatiser : Red Hat, SuSE, Mandrake. D'autre part celles, plus proches de la philosophie BSD, ne prenant pas d'initiatives et vous laissant gouverner le système : Slackware, Debian, Gentoo (listes non exhaustives).

À ce stade, deux voies sont possibles : l'administrateur maîtrise la ligne de commande d'Unix, répare son système, et la machine est rapidement remise sur pied. Deuxième possibilité, l'administrateur connaît trop mal l'interface texte, et la seule issue est alors la réinstallation complète de la machine, avec éventuellement quelques pertes de données à la clef.

Mon point de vue est que finalement, les BSD rendent un fier service à l'administrateur en l'obligeant à utiliser l'interface texte. Il y gagne sur plusieurs tableaux :

- Sa capacité à traiter efficacement des problèmes qui peuvent prendre plusieurs jours si on n'essaie de ne les traiter qu'avec l'interface graphique.
- La possibilité de prendre la main sur des machines distantes dont la connectivité est mauvaise, à cause de problèmes structurels (machine à l'autre bout du monde), ou conjoncturels (lien réseau saturé).

B.A.-BA **Modes mono-utilisateur et multi-utilisateurs**

Le mode de fonctionnement normal d'une machine Unix est le mode multi-utilisateurs, où la machine est susceptible d'accueillir en parallèle les processus de dizaines d'utilisateurs – d'où son nom.
Quand le système détecte un problème majeur au démarrage, il passe en mode mono-utilisateur (*single user*). Dans ce mode, la machine n'offre qu'une interface texte sur la console.

- La capacité à remettre en état une machine qui n'est plus capable de fonctionner en mode multi-utilisateurs.

Après avoir exposé les raisons pour lesquelles il est utile de maîtriser la ligne de commande, nous pouvons passer à son utilisation.

Où se faire la main ?

On peut se faire les dents sur le système d'un autre ou sur un système récemment installé. Si vous optez pour la deuxième solution, faites un détour par le chapitre 4, qui traite de l'installation des systèmes BSD.

Au terme de l'installation, le système propose une interface de ligne de commande, avec les droits de l'administrateur du système, l'utilisateur appelé root. Cet usager étant omnipotent, une fausse manœuvre faite en son nom peut détruire tout le système.

Apprendre la ligne de commande en tant que root vous exposera à tout réinstaller à chaque erreur grave. Il vaut donc mieux créer un compte utilisateur aux pouvoirs limités pour faire son apprentissage des commandes Unix (la création des comptes est abordée au chapitre 6). Mais pour créer ce compte, il faudra faire appel à la ligne de commande !

VOCABULAIRE **Le super-utilisateur**

Sous Unix, l'utilisateur root a les pleins pouvoirs. En anglais, on parle de *superuser* (super-utilisateur).

B.A.-BA **SSH et Telnet**

SSH signifie *Secure SHell*. C'est un protocole permettant d'avoir accès à distance à la ligne de commande d'une machine Unix, en chiffrant les communications (ce qui évite de se faire capturer son mot de passe ou d'être espionné par des indélicats). Telnet joue le même rôle que SSH, mais sans chiffrement – il est donc moins sûr.
Pour se connecter ainsi sur une machine distante, il faut disposer d'un client (Telnet ou SSH) sur la machine locale, et du serveur correspondant activé sur la machine distante. Par exemple, sous MacOS X, on active le serveur SSH dans le tableau de bord **partage** en cochant la case **Connexion à distance** dans l'onglet **Service**. Ensuite, il ne reste plus qu'à invoquer le client, taper l'adresse du serveur, et saisir son nom d'utilisateur et mot de passe.

La solution la plus simple est donc de vous faire la main chez quelqu'un d'autre. Dans le cadre professionnel, il vous est peut-être possible d'obtenir un compte sans privilège particulier sur une machine Unix. Et les universités proposent souvent des comptes Unix à leurs étudiants.

Autres pistes possibles : sur tout système MacOS X, on dispose d'un Unix ; il suffit pour cela d'invoquer le terminal ou de se connecter à distance via SSH – à condition d'y avoir un compte, bien entendu.

Dernière possibilité : certains fournisseurs de services Internet proposent des accès gratuits au shell de leur machine Unix : une requête dans votre moteur de recherche favori en indiquant « *free Unix shell* » devrait vous en donner quelques-uns. Le service fourni est en général assez limité : peu d'espace disque, pas de possibilité d'utiliser le réseau ou de compiler un programme. Ces fournisseurs de services vendent en général un service complémentaire sur la même machine pour bénéficier des fonctions inexistantes dans la prestation gratuite.

Le shell, quelques commandes sur les fichiers

Lorsque l'usager interagit avec la machine via la ligne de commande, c'est le plus souvent par le biais d'un « shell ». Le shell est un programme qui attend des commandes, les exécute, puis attend à nouveau les commandes suivantes.

Pour signaler à l'usager qu'il attend une commande, le shell affiche une invite de commande, dont la forme par défaut dépend du type de shell utilisé. Elle se

termine en général par un $ (dollar), remplacé par un # (dièse) lorsque l'usager travaille en tant qu'administrateur. Cela permet de se rappeler les pouvoirs illimités dont on dispose, et les responsabilités qui en découlent. Ces comportements correspondent aux réglages par défaut, mais il peuvent être configurés différemment. Chaque shell a aussi quelques comportements propres.

Outre le shell, un système Unix comprend un système de fichiers, organisé en arborescence en partant d'un répertoire appelé la racine, et noté / (*slash*, ou barre de division). À tout moment, le shell dispose d'un répertoire courant : l'endroit de l'arborescence sur lequel la prochaine commande agira.

Voyons quelques commandes pour naviguer dans l'arborescence. Pour les tester, il suffit de les taper à l'invite de commande, puis de valider.

- **pwd** affiche le répertoire courant, ainsi que le chemin qui y mène depuis la racine.
- **ls** affiche le contenu du répertoire courant.
- **cd rep** permet d'entrer dans le répertoire **rep**, qui devient alors le répertoire courant.
- **cd ..** est une forme particulière de la commande **cd**, qui permet de remonter dans le répertoire père du répertoire courant (celui qui contient le répertoire courant)

Il est possible de fournir à **cd** des chemins au lieu d'un simple répertoire. Par exemple, pour se rendre dans le répertoire share, situé dans répertoire usr, lui-même à la racine, on peut taper :

```
$ cd /usr/share
$ pwd
/usr/share
```

CULTURE **Les systèmes de fichiers**

Le système de fichiers Unix diffère un peu de celui du Mac (avant MacOS X) ou de Windows. En particulier, il compte une unique racine, à laquelle tout se rattache. On y trouve le contenu d'une partition (la partition racine) ; les autres partitions ou disques sont visibles dans des répertoires appelés « points de montage », tels que /home ou /cdrom.
MacOS avant sa version X et Windows ont un système de fichiers distinct pour chaque partition, tous ces volumes apparaissant soit avec leurs noms (sous MacOS), soit avec des noms de lettres comme C: ou D: (sous Windows).
Autre différence : sous Unix, les répertoires sont séparés par des caractères / (*slash*) alors que Windows utilise des \ (*backslash*) et que MacOS (avant X) utilise : (deux points).

ATTENTION **Invite de commande**

Dans l'exemple ci-contre, le $ est l'invite de commande : ne le tapez pas ! Dans tous les exemples, les commandes à taper apparaissent **comme ceci** et ce qu'affiche l'ordinateur est représenté ainsi.

HISTOIRE **Les types de shells**

Le plus ancien shell disponible s'appelle *Bourne Shell*, invoqué par la commande /bin/sh. Il est apprécié pour réaliser des scripts, car c'est le seul qui soit standardisé (norme ISO 9945-2, dite POSIX.2) Grâce à cette standardisation, son comportement ne varie pas trop d'un Unix à l'autre, ce qui est capital pour pouvoir réutiliser les scripts.
En revanche, il n'est pas très prisé pour l'utilisation interactive, car il était à l'origine peu confortable (pas de complètement, pas de rappel des commandes tapées précédemment...). Deux shells visant à le remplacer pour des sessions interactives sont donc rapidement apparus : le *Korn Shell* (**ksh**) et le *C Shell* (**csh**). Le *Korn Shell* s'efforce de reprendre la syntaxe du *Bourne Shell*, et le *C Shell* utilise une syntaxe proche du langage C.
Récemment, des évolutions de ces shells s'en sont démarquées, essentiellement pour des question de licences des logiciels : Le Bourne-Again Shell (**bash**) est un *Korn Shell* amélioré fourni par défaut sous GNU/Linux, ce qui lui a donné une forte popularité. **tcsh** est une évolution du *C Shell*.
Enfin, certains shells, comme **zsh**, tentent une synthèse entre les syntaxes du *Bourne Shell* et du *C Shell*, poussant plus loin les facilités d'utilisation interactive.
Le choix du shell interactif est avant tout une affaire de goût personnel. Pour faire des scripts, le *Bourne Shell* est difficilement contournable, puisqu'il est le seul qui soit standardisé. Le *Korn Shell* a l'avantage de permettre une utilisation interactive relativement conviviale (une fois configuré correctement), tout en gardant la syntaxe du *Bourne Shell*. Certains lui préféreront des shells plus récents et confortables, tels que **bash**.
La meilleure solution pour choisir son shell est encore d'en essayer plusieurs. Aller demander dans un forum de discussion lequel est le meilleur ne produira qu'une guerre passionnée et stérile entre les partisans de chaque shell.
La commande **echo $SHELL** permet de découvrir le shell utilisé.

Ce chemin est absolu, car il commence par un / (*slash*), ce qui signifie qu'il part de la racine du système de fichiers. On peut aussi indiquer des chemins relatifs, par exemple pour aller dans le répertoire sys du répertoire include, lui-même dans le répertoire père du répertoire courant :

```
$ pwd
/usr/share
$ cd ../include/sys
$ pwd
/usr/include/sys
```

La commande **ls** liste le contenu d'un répertoire

- **ls** s'applique au répertoire courant.
- **ls var** concerne le répertoire var.

De même qu'avec **cd**, on peut passer à **ls** des chemins plus complexes, dont voici quelques exemples :

```
$ ls /usr/share
calendar        groff_font      man             openssl         syscons
dict            info            me              pcvt            tabset
doc             isdn            misc            perl            tmac
examples        libg++          mk              sendmail        vi
games           locale          nls             skel            zoneinfo
$ cd /usr/share
$ pwd
/usr/share
$ ls
calendar        groff_font      man             openssl         syscons
dict            info            me              pcvt            tabset
doc             isdn            misc            perl            tmac
examples        libg++          mk              sendmail        vi
games           locale          nls             skel            zoneinfo
$ ls ../../var
account         cron            games           mail            rwho
at              db              heimdal         msgs            spool
```

CULTURE **Organisation du système de fichiers Unix**

Lors de l'exploration du système de fichiers, vous vous interrogerez peut-être sur la fonction de chacun des répertoires. Voici celle de quelques répertoires courants :

/etc	Fichiers de configuration.
/home	Répertoires personnels des utilisateurs.
/dev	Pseudo-fichiers donnant accès au matériel.
/var	Fichiers de taille variable, comme par exemple les journaux, ou les files d'attente de messagerie et d'impression.
/bin	Commandes utilisateur indispensables à la maintenance de la machine.
/sbin	Commandes d'administration indispensables à la maintenance de la machine.
/usr	Tout ce qui n'est pas indispensable à la maintenance de la machine, mais sert lors de son fonctionnement normal.
/usr/bin	Les autres commandes utilisateur.
/usr/sbin	Les autres commandes d'administration.
/usr/local	Arborescence réservée à l'administrateur. Le système n'y installe jamais rien.
/tmp	Fichiers temporaires, effacés à chaque redémarrage.
/var/tmp	Fichiers temporaires, préservés entre les redémarrages.

La documentation comprend une description exhaustive du système de fichiers, qu'on peut consulter en tapant **man hier** à l'invite de commande. Nous détaillerons sous peu l'utilisation de cette aide en ligne ; sachez d'ores et déjà que la touche **Espace** permet d'avancer d'un écran et que la touche **q** quitte l'aide en ligne.

```
backups         dnews           log             preserve        tmp
crash           empty           lost+found      run             yp
$ cd /
$ pwd
/
$ ls var
account         cron            games           mail            rwho
at              db              heimdal         msgs            spool
backups         dnews           log             preserve        tmp
crash           empty           lost+found      run             yp
```

La commande **ls** admet des options. Dans les commandes Unix, les options prennent la forme d'une ou plusieurs lettres, précédées par un **–**. Par exemple, l'option **–a** (comme *all*) de **ls** permet de lister tous les fichiers, y compris les fichiers « cachés » :

```
$ ls /tmp
truc.txt
$ ls -a /tmp
.               ..              .X11-unix       truc.txt
```

De même, l'option **–l** (comme *long*) permet de voir plus d'informations sur chaque fichier. Voici des exemples de listes fournies par **ls –l** et **ls –a –l** :

```
$ ls -l /
total 4816
drwxr-xr-x   2 root  wheel      512 Jul 13 22:24 altroot
drwxr-xr-x   2 root  wheel     1024 Aug 27 03:22 bin
drwxr-xr-x   4 root  wheel    10752 Sep 14 21:50 dev
drwxr-xr-x  19 root  wheel     2048 Oct 22 21:47 etc
drwxr-xr-x   4 root  wheel      512 Aug 27 06:51 home
drwxr-xr-x   2 root  wheel      512 Jul 11 12:51 mnt
-rw-r--r--   1 root  wheel  2410055 Aug 31 09:42 netbsd
drwxr-xr-x   4 root  wheel      512 Aug 28 18:39 root
drwxr-xr-x   2 root  wheel     2048 Jul 12 23:27 sbin
drwxr-xr-x   2 root  wheel      512 Jul 11 12:51 stand
lrwxr-xr-x   1 root  wheel       11 Jul 13 11:29 sys -> usr/src/sys
drwxrwxrwt   3 root  wheel      512 Oct 24 12:34 tmp
drwxr-xr-x  15 root  wheel      512 Aug 28 11:34 usr
drwxr-xr-x  23 root  wheel      512 Jul 16 15:11 var
$ ls -l -a /tmp
total 68
drwxrwxrwt   3 root  wheel    512 Mar 15 04:34 .
drwxr-xr-x  20 root  wheel    512 Mar  6 10:37 ..
drwxrwxrwt   2 root  wheel    512 Mar  6 10:51 .X11-unix
-rw-r--r--   1 root  wheel  51200 Feb  8 16:27 truc.txt
```

On observe dans ces exemples la liste détaillée des fichiers et répertoires présents à la racine. Décrivons-la :

Dans la première colonne, on trouve une première lettre indiquant le type de l'objet considéré. Voici les types les plus courants :

- `–` fichier normal
- `d` répertoire
- `l` lien symbolique (équivalent des alias sur Mac et des raccourcis sous Windows)

Les autre caractères indiquent les droits sur le fichier (voir la note ci-contre).

La deuxième colonne indique le nombre de références existant sur l'objet. En l'absence de liens durs (voir l'aparté), ce nombre est égal à 1 pour un fichier, et au nombre d'éléments contenus pour un répertoire.

ASTUCE **Les fichiers cachés**

Les fichiers et répertoires dont le nom commence par un . (point) sont invisibles lorsque l'on utilise **ls** (sauf pour l'utilisateur root, qui voit toujours tout). **ls –a** permet de voir tous les fichiers et répertoires. Cela permet d'alléger la présentation en manipulation courante, ces fichiers étant usuellement des fichiers de configuration qu'on ne souhaite pas voir apparaître en permanence.
Chaque répertoire contient au moins deux éléments dont le nom commence par un . (point) : le répertoire . (point), qui désigne le répertoire courant, et le répertoire .. (point point), qui désigne le répertoire père du répertoire courant. Le répertoire racine est son propre père.
Le répertoire .. (point point) est d'utilisation courante ; nous en avons déjà vu un exemple. On utilise le répertoire . (point) dans certains cas plus rares ; nous en verrons un plus loin.

EN PRATIQUE **L'affichage des droits**

La figure 3.1 décrit l'attribution des droits affichés par **ls**. Un `r` indique un droit en lecture (*read*), un `w` un droit en écriture (*write*), et un `x` un droit en exécution. Un – signale une absence de droit.

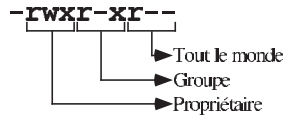


Figure 3–1 L'affichage des droits

PLUS LOIN **Autres types d'objets**

Il existe d'autres types d'objets, qui ne sont pas des vrais fichiers, et qui ne vous intéresseront pas immédiatement. Pour les nommer rapidement :

- `c` périphériques en mode caractères
- `b` périphériques en mode blocs
- `p` tubes nommés (*named pipes*, ou FIFO)
- `s` point d'entrée de protocoles de réseau locaux à la machine (*sockets* UNIX)

PLUS LOIN **Taille des répertoires**

Un répertoire est en quelque sorte un fichier contenant la liste des éléments qu'il renferme. C'est la taille (en octets) de ces informations qui est indiquée dans la cinquième colonne du résultat de **ls -l**. Pour obtenir la véritable taille (en kilo-octets) d'un répertoire, utilisez la commande **du -ks rep**, où **rep** est le nom du répertoire concerné.

Les troisième et quatrième colonnes indiquent respectivement l'utilisateur et le groupe propriétaires du fichier. Nous aborderons les groupes d'utilisateurs dans le détail au chapitre 6.

Le cinquième champ contient la taille de l'objet en octets. Pour les fichiers, il s'agit de la taille de leur contenu ; pour les autres objets, c'est plus complexe.

Viennent ensuite la date et l'heure de dernière modification du fichier, et enfin le nom du fichier lui-même. Le nom des liens symboliques est suivi du chemin de l'objet cible.

À l'aide !

La commande **ls** compte encore beaucoup d'options. Où trouver leur liste ? Comment interpréter la colonne donnant les droits d'accès aux fichiers ?

La réponse à ce type d'interrogations légitimes est donnée par la commande **man**, dont le rôle est d'afficher la documentation (page **man**) d'une commande donnée. Exemple pour la commande **ls** :

```
$ man ls
LS(1)                      NetBSD Reference Manual                      LS(1)

NAME
     ls - list directory contents


(...)
```

Les pages **man** sont nombreuses, exhaustives et souvent bien à jour. C'est sans doute la meilleure source de documentation que l'on puisse trouver sur un système Unix. Elles sont également difficiles à lire, austères, et en anglais.

PLUS LOIN **Liens symboliques, liens durs, et nombre de références**

Il y a deux types de liens : les liens symboliques et les liens durs. Un **ls -l** ne montre les destinations que pour les liens symboliques, petits fichiers contenant l'emplacement du fichier destination – qui peut éventuellement ne pas exister. On crée les liens symboliques avec la commande **ln -s** :

```
$ ln -s /etc/profile test
$ ls -l
total 0
lrwxr--r-- 1 manu  wheel  12 Jul 21 16:18 test -> /etc/profile
```

Les liens durs, qui se créent en utilisant la commande **ln** sans l'option **-s**, consistent à faire pointer plusieurs noms de fichiers vers le même *inode*. Les *inodes* sont des structures internes utilisées par le système pour gérer les objets présents dans le système de fichiers : chaque fichier ou répertoire est associé à un et un seul *inode*.
ls -l ne montre pas la destination d'un lien dur, car il n'y a pas de nom de fichier destination. Il y a simplement plusieurs noms de fichier donnant accès au même contenu. Par contre, les liens durs sont décelables en observant la deuxième colonne de l'affichage de **ls -l**, qui indique le nombre de références vers un objet. Pour un fichier, il est égal au nombre de noms pointant sur le même *inode*. S'il est supérieur à 1, il y a un lien dur.

```
$ ls -l
total 2
-rw-r--r--  1 manu  wheel  6 Jul 21 16:26 test
$ ln test nom2
$ ls -l
total 4
-rw-r--r--  2 manu  wheel  6 Jul 21 16:26 nom2
-rw-r--r--  2 manu  wheel  6 Jul 21 16:26 test
$ ln test nom3
$ ls -l
total 6
-rw-r--r--  3 manu  wheel  6 Jul 21 16:26 nom2
-rw-r--r--  3 manu  wheel  6 Jul 21 16:26 nom3
-rw-r--r--  3 manu  wheel  6 Jul 21 16:26 test
```

Il est important de signaler qu'on ne peut faire des liens durs qu'entre des objets présents sur la même partition.

Détaillons :

- Difficiles à lire : c'est vrai, du moins au début. Utiliser efficacement cette documentation suppose un certain apprentissage. Nous donnerons plus loin quelques clefs pour mieux digérer les pages **man**.
- Austères : c'est encore vrai. Tout comme d'ailleurs le reste de ces systèmes BSD. Il faut prendre les choses du bon côté : lorsque vous prenez place au volant d'une formule 1, vous ne vous attendez pas à trouver un allume-cigare ou un autoradio, mais la formule 1 a quand même certains plus par rapport à une Lada.
- En anglais : qu'on le déplore ou qu'on s'en réjouisse l'anglais est la langue internationale pour la plupart des disciplines techniques, et en particulier pour l'informatique. La documentation est donc d'abord écrite en anglais puis parfois traduite. Qui refuse d'apprendre l'anglais technique, risque de se priver d'une partie de la documentation, ou se de limiter à des versions dépassées.

  Sur les systèmes BSD, les pages **man** ne sont pas traduites, plus par manque de moyens humains que par choix politique. Si lire l'anglais technique représente un obstacle insurmontable, les BSD ne sont probablement pas faits pour vous.

Revenons sur les pages **man** proprement dites. Toutes les pages documentant des commandes ont la même structure, dont il faut s'imprégner pour être efficace dans sa lecture.

On trouve dans chaque page le nom de la commande, suivi entre parenthèses de la section du manuel dans laquelle elle se trouve. Il y a 8 sections sur un Unix traditionnel, et parfois quelques sections supplémentaires, spécifiques au système. Par exemple, les BSD ont une section 9 qui documente le noyau, certains systèmes Unix commerciaux ont une section 3F pour les interfaces de programmation en Fortran, etc.

- section 1 : commandes pour l'utilisateur
- section 2 : appels système, interfaces pour les programmes en C

PLUS LOIN **La visionneuse**

Les pages **man** sont affichées à l'aide d'une commande appelée visionneuse (*pager* en anglais). C'est par défaut **less**, ou **more** sur les Unix anciens. Son rôle est de présenter la documentation écran par écran : sur un terminal de 25 lignes, elle affiche 24 lignes, puis attend une pression sur la touche **Espace**, qui affichera les 24 lignes suivantes, et ainsi de suite.
Avec **less**, on peut remonter ou redescendre avec les touches fléchées. Avec **more**, la touche **u** (*up*) permet de monter, la touche **entrée** de descendre. À tout moment, on peut quitter **less** ou **more** en enfonçant la touche **q** (quitter).

CULTURE **Pages man et pages info**

Les pages **man** ne sont hélas pas universelles. La *Free Software Foundation*, à l'origine de beaucoup de programmes pour Linux et BSD (et en particulier de tous les outils de compilation) a décidé que sa documentation serait produite en pages **info** plutôt qu'en pages **man**. Les pages **info** s'utilisent comme les pages **man**, mais avec la commande **info**. La navigation y est assez différente.

```
$ info gcc
File: gcc.info,  Node: Top,  Next: G++ and GCC,  Up: (DIR)

Introduction
************

   This manual documents how to run, install and port the GNU compiler,

(...)
```

Certains volontaires écrivent des pages **man** pour les programmes de la FSF, mais elles ne sont pas toujours très à jour. Ce sont les pages **info** qui font autorité pour ces programmes.

- section 3 : fonctions des bibliothèques du langage C
- section 4 : points d'entrée du noyau, pilotes
- section 5 : fichiers de configuration
- section 6 : jeux
- section 7 : standards
- section 8 : commandes d'administration
- section 9 : fonctions présentes dans le noyau

Certaines pages **man** existent dans plusieurs sections : on trouve par exemple une commande **passwd** (pour changer son mot de passe) et un fichier de configuration passwd (base de données des utilisateurs existant sur le système). Il y a donc deux pages **man**, en sections 1 et 5. Si l'on tape **man passwd**, la commande **man** affichera la première page trouvée, en section 1. Pour obtenir la page en section 5, il faut la demander explicitement : **man 5 passwd**.

EN PRATIQUE **Plusieurs pages au même nom**

Comment savoir s'il y a des pages de même nom dans d'autres sections ? On peut tout simplement demander à la commande **man** d'aller chercher la page dans toutes les sections une par une. On peut aussi jeter un œil à la rubrique SEE ALSO de la première page trouvée : elle précise la plupart du temps s'il y a d'autres pages **man** de même nom dans d'autres sections du manuel.

SUR LES AUTRES UNIX **Choisir la section**

Sur certains systèmes Unix, on sélectionne la section du manuel avec l'option **-s** : **man -s 5 passwd**

La structure des pages man

Pour une section donnée, les pages **man** respectent toujours la même structure. Si la lecture est un peu pénible les premières fois, on gagne en efficacité une fois que l'on sait où chercher l'information. Voici la structure des pages pour les sections 1 et 8 (les commandes) :

- NAME : nom de la commande
- SYNOPSYS : mode d'emploi synthétique de la commande. La notation est toujours la même : les éléments optionnels entre crochets. les choix possibles séparés par un caractère | (barre verticale), etc.
- DESCRIPTION : description complète de la commande, comprenant notamment le détail de chaque option disponible. La liste des options est une portion fréquemment consultée.
- SEE ALSO : les autres pages **man** peu ou prou liées à celle-ci. Cette rubrique est importante car elle permet de trouver une commande dont on ne connaît pas encore le nom.

On trouve aussi quelques sections optionnelles : ENVIRONMENT, STANDARD, HISTORY, COMPATIBILITY, pas présentes dans toutes les pages.

Lorsque l'on consulte une page **man**, on peut chercher un mot en tapant **/** (*slash*) suivi d'une portion significative de ce mot, et validation. En appuyant ensuite sur

EN PRATIQUE **Le synopsis**

Quelques exemples de synopsis valent mieux qu'un long discours. Le synopsis (syntaxe) indique comment utiliser une commande :

- cmd [-ab] indique que l'on peut faire **cmd**, **cmd -a**, **cmd -b** ou encore **cmd -a -b**.
- cmd -b [-a] donne les possibilités **cmd -b -a** ou **cmd -b**.
- cmd [-a|-b] autorise à utiliser **cmd**, **cmd -a** ou **cmd -b**.

L'ordre des options est en général sans importance, et on peut compacter ces dernières comme suit : **cmd -ab** représente **cmd -a -b**

n, on passe à l'occurrence suivante du mot ; **N**, permet de passer à l'occurrence précédente. La recherche de mots-clef dans une page **man** est un véritable réflexe à prendre pour lire plus efficacement. Imaginons que vous recherchiez l'option de **ls** qui permet d'afficher une barre de division (*slash*) à la fin de chaque répertoire pour les distinguer des autres éléments. Sur un système BSD, rechercher le mot « *Directory* » dans la page **man** de **ls** vous livrera la réponse en quelques secondes : **ls -p**.

ASTUCE **Chercher sans la casse**

Les mots en début de phrase commençant par des majuscules, il faut penser à chercher avec et sans majuscule initiale, ou omettre la première lettre : taper **irectory** au lieu de **directory**.

Trouver la bonne page man

Dernier point sur la documentation : comment trouver une commande sans connaître son nom ? Les pages **man** n'aident pas tellement dans un tel cas de figure. On peut essayer de rebondir de page en page avec la rubrique SEE ALSO, mais ça n'est pas toujours possible. Par exemple, comment retrouver les commandes qui permettent d'utiliser un modem ?

C'est l'objet de la commande **apropos** :

```
$ apropos modem
chat (8) - Automated conversational script with a modem
mhzc (4) - Megahertz Ethernet/modem card driver
umodem (4) - USB modem support
```

SUR LES AUTRES UNIX **apropos ou pas ?**

Sur certains Unix, **apropos** n'existe pas ; il faut remplacer cette commande par **man -k**.

CULTURE **Désignation des pages man**

Pour préciser que l'on parle d'une page **man**, on indique souvent le nom de la commande suivie de la section de manuel dans laquelle on peut la trouver, mis entre parenthèses. Exemples : ls(1), pwd_mkdb(8), wtf(6).

Cela fournit une seule commande (en section 8). La lecture de chat(8) et des autres pages listées dans la section SEE ALSO devrait permettre de découvrir peu à peu comment configurer une connexion PPP via un modem.

Plus de commandes portant sur les fichiers

Nous avons vu comment se déplacer dans l'arborescence et lister des fichiers. Voici d'autres commandes fondamentales pour la manipulation des fichiers et des répertoires, accompagnées de quelques exemples d'utilisation :

Copier :

```
$ cp /etc/services /tmp/services
$ cd /tmp
$ ls
services
$ cp services services.bak
$ ls
services        services.bak
$ cp /etc/services /etc/protocols /etc/hosts /tmp
$ ls
hosts           protocols       services        services.bak
```

Déplacer :

```
$ mv /tmp/services /var/tmp/services
$ cd /var/tmp
$ ls
services
$ mv /tmp/protocols /tmp/hosts ./
$ ls
hosts      protocols  services
```

```
$ mv protocols services hosts /tmp
$ ls ./ /tmp
./:

/tmp:
hosts           protocols       services        services.bak
```

Changer de nom (une forme de déplacement) :

```
$ mv /tmp/protocols protocols.old
$ ls
protocols.old
$ mv protocols.old protocols2.old
$ ls
protocols2.old
$ mv protocols2.old /tmp/protocols
$ ls
$
```

Effacer :

```
$ cd /tmp
$ ls
hosts           protocols       services        services.bak
$ rm services.bak
$ ls
hosts           protocols       services
$ cd /var
$ rm /tmp/services /tmp/hosts
$ ls /tmp
protocols
$ rm ../tmp/protocols
$ ls /tmp
$
```

Créer un répertoire ou le supprimer :

```
$ cd /tmp
$ ls -p
$ mkdir bidule
$ ls -p
bidule/
$ mkdir bidule/truc
$ ls -p bidule
truc/
$ mkdir -p /tmp/test/foo/bar/buz
$ ls -p
bidule/         test/
$ rm -R /tmp/test
$ ls -p
bidule/
$
```

Et ainsi de suite... Tout cela est assez monotone, mais il faut bien un vocabulaire minimal avant de passer à des choses plus compliquées. Vous découvrirez les sens exact des options **`-p`** de **`mkdir`**, **`-p`** de **`ls`**, et **`-R`** de **`rm`** dans leurs documentations respectives.

ATTENTION **Pas de corbeille !**

Sous Unix, on a de l'hygiène : on ne fouille pas dans les poubelles. Tout fichier effacé est détruit immédiatement et pour toujours.
Pour reproduire le comportement de la corbeille de MacOS ou Windows, il faudra déplacer les fichiers dans un répertoire poubelle au lieu de les supprimer avec la commande **`rm`**.

L'éditeur `vi`

La configuration d'un système Unix est enregistrée dans un certain nombre de fichiers, en général de simples fichiers texte, qui se trouvent pour la plupart dans le répertoire /etc. Nous avons examiné différentes commandes pour manipuler des fichiers, mais tout ceci sera de peu d'utilité sans la maîtrise d'un outil permettant de modifier le contenu des fichiers : un éditeur de textes.

Il existe de nombreux éditeurs de textes sous Unix. Certains d'entre eux fonctionnent avec une interface graphique et ressemblent au bloc-notes de Windows ou à SimpleText sur Mac. Notre but étant la maîtrise de l'interface texte, nous ne les aborderons pas.

Citons certains éditeurs classiques en mode texte : **`ed`**, **`vi`**, **`emacs`**, **`pico`**, **`jed`**, **`vim`**... Les plus répandus sont sans conteste **`vi`** et **`emacs`**. Ils existent sur beaucoup de systèmes Unix, et disposent chacun de leur propre communauté d'utilisateurs. La question du meilleur éditeur est un grand classique sur lequel les adeptes de **`vi`** et **`emacs`** aiment à débattre à l'infini sur Usenet.

Les autres sont tous plus récents et on les trouve plus rarement sur des systèmes Unix trop anciens. **`vim`** est un **`vi`** amélioré (*Vi IMproved*), **`pico`** est l'éditeur fourni avec le logiciel de messagerie en mode texte **`pine`**. Certains éditeurs sont capables de fonctionner en mode graphique le cas échéant : **`gvim`** est la version graphique de **`vim`** ; **`emacs`** et **`xemacs`** peuvent fonctionner à la fois en mode texte et en mode graphique.

On n'a donc que l'embarras du choix. Fort bien, mais quel éditeur choisir ? Cette question reste ouverte pour le novice, auquel on conseillera d'essayer tous les éditeurs mis à sa disposition pour décider lequel lui convient le mieux. Pour l'administrateur, le choix est plus restreint : l'éditeur à apprendre, c'est celui qui est disponible juste après l'installation du système, et avant la configuration de quoi que ce soit d'autre. Sur beaucoup de systèmes et en particulier sur les systèmes BSD, ce choix se limite à **`vi`**.

CULTURE **Usenet**

Usenet est un système distribué de groupes de discussions. Vous en avez peut-être entendu parler sous le nom « les *news* ». C'est un service gratuit qui vous permettra de discuter de tout et de n'importe quoi avec des interlocuteurs du monde entier.
Pour y accéder, il faut disposer d'un logiciel de *news* (KNews, slrn, Netscape, MacSOUP, FreeAgent, WinVN... les clients ne manquent pas), et de l'adresse du serveur de *news* de votre fournisseur d'accès.

VOCABULAIRE **`vi`**

`vi` tient son nom de *VIsual editor*. En anglais, **`vi`** se prononce « vie ail ».

CONFORT **`ee` sous FreeBSD**

Si vous travaillez sous FreeBSD, vous disposez dans la distribution de base d'un éditeur simple à utiliser en plus du traditionnel **`vi`** : **`ee`**.

Une première utilisation de `vi`

Pour démarrer **`vi`**, tapez dans le shell **`vi`** suivi du nom du fichier à éditer, puis validez. On peut aussi invoquer **`vi`** sans indiquer de nom de fichier, auquel cas l'éditeur créera un nouveau fichier, mais il faudra lui indiquer où l'enregistrer au moment de quitter.

```
$ vi /etc/hosts
```

Le shell laisse le terminal à **`vi`** et vous obtenez un écran comme celui-ci :

```
#       $NetBSD: hosts,v 1.6 2000/08/15 09:33:05 itojun Exp $
#
# Host Database
# This file should contain the addresses and aliases
# for local hosts that share this file.
# It is used only for "ifconfig" and other operations
# before the nameserver is started.
#
```

ASTUCES **Si cela ne fonctionne pas**

Si **`vi`** refuse de démarrer, c'est peut-être la variable `TERM`, qui indique le type de terminal, qui est à mettre en cause. Voici un réglage qui convient souvent **`TERM=vt100; export TERM`** en **`sh`**, **`ksh`** ou **`bash`**, ou **`setenv TERM vt100`** en **`csh`** ou **`tcsh`**.
Autres causes d'échec : les répertoires /tmp ou /var/tmp ne sont pas accessibles en écriture. C'est le cas de certaines installations inachevées. Voyez le chapitre 5 pour trouver des solutions à ces problèmes.

```
#
::1                     localhost
127.0.0.1               localhost
#
# RFC 1918 specifies that these networks are "internal".
# 10.0.0.0      10.255.255.255
# 172.16.0.0    172.31.255.255
# 192.168.0.0   192.168.255.255
~
~
~
~
~
~
~
~
/etc/hosts: unmodified, readonly: line 1
```

vi a deux modes de fonctionnement : un mode commande (dans lequel il se place au démarrage) et un mode insertion. En mode commande, **vi** attend des commandes, qui consistent en un ou plusieurs caractères (qui n'apparaîtront pas à l'écran).

Familiarisez-vous avec les déplacements : **h**, **j**, **k** et **l**, qui servent respectivement à déplacer le curseur d'un caractère vers la gauche, le bas, le haut, et la droite.

Le curseur ne se déplace pas toujours exactement comme attendu. En particulier, il ne peut pas atteindre une position située au-delà de la fin d'une ligne, et il se place à la fin des tabulations. Vous pouvez expérimenter ce comportement dans le fichier /etc/hosts : celui des BSD comprend des tabulations dans les lignes décrivant les adresses internes.

Pour taper du texte, il faut passer en mode insertion. Voici certaines des commandes disponibles pour cela (dans un premier temps seules les deux premières vont vous intéresser).

- **i** (*insert*) passe en mode insertion à gauche du curseur.
- **a** (*append*) passe en mode insertion à droite du curseur.
- **o** crée une ligne vide sous la ligne actuelle et passe en mode insertion au début de cette nouvelle ligne.
- **O** de même, au-dessus du curseur.

En mode insertion, tout ce que vous tapez s'affiche à l'écran. Pour se déplacer ou effectuer une quelconque commande, il faudra repasser en mode commande, en enfonçant la touche d'échappement **Échap** .

x est une commande très utile : elle efface le caractère situé sous le curseur. Sans elle, difficile de corriger ses erreurs, dans la mesure où l'on ne peut effacer que ce que l'on vient de taper lorsqu'on est en mode insertion.

CONFORT **Les touches fléchées**

En général, les touches fléchées du clavier permettent aussi de se déplacer, mais elles ne fonctionneront pas sur certains terminaux mal configurés. Il en va de même pour le pavé numérique.

EN CAS DE COUP DUR **Pas de touche Échap ?**

Certains anciens terminaux n'ont pas de touche d'échappement, mais on peut la simuler avec la combinaison **Control**+**[**.

ASTUCE **Pour connaître sa position**

Quand on ne sait plus dans quel mode on se trouve, voici une méthode simple : taper deux fois sur la touche d'échappement **Échap**. Cela a pour effet de revenir en mode commande ou d'annuler toute éventuelle commande complexe, de plusieurs caractères, qui serait en cours de saisie.

PIÈGE À C... **La casse compte**

Attention, la casse des commandes compte. Si **x** minuscule efface le caractère sur lequel est le curseur, **X** majuscule efface le caractère à sa gauche et déplace le curseur d'un cran à gauche. Vous constaterez que **vi** devient très déroutant quand on a verrouillé par mégarde les majuscules (touche **Maj**).

Quitter vi et enregistrer

Si vous avez vraiment pratiqué au fur et à mesure que vous lisiez, vous devez commencer à détester **vi**. Ce fut notre lot commun ; voyons donc comment quitter **vi**.

En mode commande, si vous tapez **:** (deux points), deux points s'affichent en bas de l'écran. **vi** attend une commande longue, et il affichera ce que vous tapez en

bas de l'écran. Une fois la commande terminée, vous devez la valider en pressant sur **Entrée**. Voici les commandes pour enregistrer et sortir :

- **:q** Tente de quitter **vi** (échouera si un fichier en cours d'édition dispose de modifications non enregistrées).
- **:w** Enregistre les modifications.
- **:wq** Enregistre et quitte **vi**.
- **:q!** Quitte **vi** sans rien enregistrer.
- **:w!** Force l'enregistrement (utile pour les fichiers protégés en écriture mais pour lesquels on peut outrepasser cette protection).
- **:w /tmp/hosts.copy** Enregistre sous le nom /tmp/hosts.copy. Cette commande sert notamment quand on a invoqué **vi** sans indiquer de nom de fichier.
- **:e /etc/protocols** Ouvre le fichier /etc/protocols. Cette commande sera rejetée si le fichier en cours de manipulation n'est pas mis à jour sur le disque.

vi agit comme n'importe quel éditeur de textes : il refuse de quitter si toutes les modifications ne sont pas enregistrées. On constate que la ligne de saisie des commandes longues propose toutes les commandes habituelles du menu **Fichier** des éditeurs de textes à interface graphique : **ouvrir**, **enregistrer**, **enregistrer sous**, **quitter**.

Plus loin avec vi

Sous ses dehors abrupts, **vi** est un éditeur de texte très complet, et très apprécié de ceux qui en ont pris l'habitude. Voici quelques commandes utiles :

Annulation, répétition

On peut annuler dans **vi** : ceci se fait avec la commande **u** (*undo*). La répétition de la commande précédente est aussi possible, avec la commande **.** (point).

Numéros de lignes

Parfois un message d'erreur vous indiquera un problème à une ligne donnée d'un fichier de configuration. Dans cette situation, vous trouverez la commande longue **:set ruler** utile : elle vous donnera votre position (colonne, ligne) dans le fichier.

ATTENTION **Overdose de vi**

Si **vi** vous échauffe déjà les méninges, passez à la section suivante, vous en savez déjà assez pour manipuler des fichiers de configuration. Retenez juste que l'on peut annuler, copier/coller, ou rechercher/remplacer. Vous reviendrez ici lorsque vous aurez besoin de ces commandes.

ASTUCES **Autres commandes :set utiles**

Sur une connexion à forte latence, on constate parfois des problèmes avec les touches fléchées. Elles envoient en effet des codes de plusieurs caractères. Quand le dernier arrive trop longtemps après le premier, **vi** considère que cela n'est plus la même commande. Pour régler cela, **:set notimeout**.

Si vous copiez/collez dans **vi** avec des commandes copier/coller externes à **vi** (copier/coller depuis un autre système d'exploitation dans une session SSH d'un XTerm, par exemple), vous serez parfois ennuyé par les indentations automatiques. Pour les supprimer, **:set noai** (*no auto-indent*).

CULTURE **Le caractère $**

Dans les commandes de **vi**, et plus généralement dans beaucoup de programmes Unix, **$** dénote la fin d'une ligne ou d'un fichier. De même, **^** (accent circonflexe) indique souvent un début.

Pour se déplacer rapidement, on peut taper un nombre suivi de la lettre **G**. Cette commande amène au numéro de ligne correspondant. Ainsi, **1G** amène au début du document, et **$G** (ou **G**) vous envoie à la fin.

On peut aussi placer des signets dans le document. À tout moment, la commande longue **:ma a** (ou **ma**) place le signet **a** à la ligne courante (on peut utiliser les 26 lettres majuscules et les 26 lettres minuscules pour les noms de signets). En mode commande, **'a** (apostrophe **a**) amène au signet **a**

Couper, copier, coller

Les copier/coller simples se font ligne par ligne. Il y a pour cela trois commandes : **dd** coupe la ligne courante, **yy** la copie et **p** colle ce qui a été coupé ou copié. Précédées d'un nombre, ces commandes porteront sur le nombre d'occurrences ainsi précisé (nombre de lignes ou de reproductions). Par exemple, **4yy** copie quatre lignes à partir de la ligne courante.

Les commandes copier et coller sont aussi disponibles en commandes longues, sous la forme **:d** et **:y**. On peut les utiliser pour indiquer des zones de fichier plus complexes. Par exemple, **:d** va couper la ligne courante, et **:3,5d** va couper les lignes 3 à 5. **:1,$y** copie l'ensemble du fichier (n'oubliez pas : le dollar représente la fin). **:-1,+3d** coupe à partir de la ligne précédant la ligne courante et jusqu'à la 3ème ligne après la ligne courante. Dernier exemple : **:2,.d** coupe tout de la deuxième ligne à la ligne courante (le **.** (point) représente le numéro de la ligne courante).

On peut aussi utiliser les signets : **:'a,'by** copie les lignes situées entre les signets **a** et **b** ; c'est très pratique.

Rechercher et remplacer

Taper **/** (barre de division), passe **vi** en mode commande longue, et ce caractère s'affiche en bas de l'écran. Tapez un texte à rechercher et validez. **n** passe à l'occurrence suivante, et **N** à l'occurrence précédente. C'est le même comportement que dans les visionneuses classiques **less** et **more**, qui sont utilisées pour les pages **man**.

PLUS LOIN **Les expressions régulières**

Pour approfondir ses connaissances sur les expressions régulières et leur utilisation, le lecteur peut se rapporter à l'ouvrage suivant, référence reconnue dans le domaine : *Maîtrise des expressions régulières*, de Jeffrey E. F. Friedl.
À noter un débat d'expert : certaines personnes plaident pour la traduction de l'anglais « *regular expression* » par « expressions rationnelles », plutôt que « expressions régulières ». Vous pouvez donc être amené à trouver ce terme, qui reste minoritairement employé face à « expressions régulières ».

La commande rechercher/remplacer est un peu complexe. Elle fonctionne comme les couper et copier en commande longue : **:1,$s/foo/bar/** remplace **foo** par **bar** dans tout le fichier. **:'a,'bs/foo/bar/** se contente de remplacer entre les signets **a** et **b**. **:.,+3s/foo/bar/** fait le rechercher/remplacer entre la ligne courante et 3 lignes plus loin.

Par défaut, **vi** ne fait qu'un remplacement par ligne. Pour remplacer plusieurs occurrences sur une même ligne, il faut ajouter le modificateur **g** (global) à la fin de la commande : **:1,$s/foo/bar/g**. Le modificateur **c** demande confirmation pour chaque substitution ; on peut alors choisir **y** (oui), **n** (non), **a** (toutes : *all*) ou **q** (quitter).

Le rechercher/remplacer de **vi** permet des constructions extrêmement complexes à l'aide d'« expressions régulières », langage symbolique permettant de définir

des familles de mots. Il y a beaucoup à dire en la matière, mais trop pour en parler en détail dans ce chapitre. En attendant de devenir un expert, il faut savoir que certains caractères (comme le point) ont un sens particulier dans une recherche pour **vi**. Ainsi, le point correspond à n'importe quel caractère. Pour mentionner un véritable point, il faudra le précéder de **** (*backslash*) comme suit : **\.** (*backslash* point).

Commandes externes

vi est riche de nombreuses possibilités. L'énumération précédente peut décourager, mais il ne faut pas s'inquiéter. Le tout est d'être capable d'éditer des fichiers ; le reste n'est qu'une histoire d'efficacité dans l'utilisation des outils. Très peu de personnes connaissent toutes les commandes de **vi** !

Parfois, malgré ce foisonnement de commandes, on a un besoin que **vi** ne sait pas traiter. Pas de problème dans ce cas, car **vi** sait déléguer à une commande externe. La commande suivante fait appel à **awk** (traitée plus loin) pour échanger les deux colonnes d'un fichier : **:1,$!awk '{print $2,$1}'**

AU SECOURS ! **Je ne supporte déjà plus vi**

vi est indispensable lors de l'installation de la machine et dans les situations critiques, mais en temps normal, vous pouvez tout à fait utiliser un éditeur plus convivial. Nous verrons au chapitre 11 comment installer des éditeurs qui paraîtront plus agréables à utiliser au débutant.

Et si on n'a pas vi ?

Vous aurez peut-être affaire à une situation où **vi** n'est pas disponible. Dans ce cas, il faudra peut-être vous résoudre à utiliser un démon de l'ancien monde : **ed**.

ed est un éditeur adapté aux terminaux imprimante : sans écran, pas d'affichage pleine page, mais ligne à ligne. Vous tapez des commandes, **ed** répond, et il n'est jamais possible de se déplacer dans le document.

ed fonctionne un peu comme **vi**. Il démarre en mode commande. On accède au mode insertion avec les commandes **a** (passe en mode insertion sous la ligne indiquée), **i** (de même, mais au-dessus), et **c** (change la ligne indiquée). Pour ressortir du mode insertion, il faut taper un **.** (point) seul sur une ligne, suivi de la touche **Entrée**.

Voici un exemple de session avec **ed** (à l'ouverture et à l'enregistrement, **ed** indique la taille du fichier : 782 puis 834 octets) :

```
$ ed /tmp/passwd
782
1,5l
root:*:0:0:Charlie &:/root:/bin/ksh
toor:*:0:0:Bourne-again Superuser:/root:/bin/sh
daemon:*:1:31:The devil himself:/:/sbin/nologin
operator:*:2:5:System &:/usr/guest/operator:/sbin/nologin
bin:*:3:7:Binaries Commands and Source:/:/sbin/nologin
2a
insertion après la ligne
numéro 2
.
1,5l
root:*:0:0:Charlie &:/root:/bin/ksh
toor:*:0:0:Bourne-again Superuser:/root:/bin/sh
insertion après la ligne
numéro 2
daemon:*:1:31:The devil himself:/:/sbin/nologin
3d
1,5l
root:*:0:0:Charlie &:/root:/bin/ksh
toor:*:0:0:Bourne-again Superuser:/root:/bin/sh
```

```
numéro 2
daemon:*:1:31:The devil himself:/:/sbin/nologin
operator:*:2:5:System &:/usr/guest/operator:/sbin/nologin
3c
je modifie la ligne 3
Et j'en ajoute une en-dessous
.
1,5l
root:*:0:0:Charlie &:/root:/bin/ksh
toor:*:0:0:Bourne-again Superuser:/root:/bin/sh
je modifie la ligne 3
Et j'en ajoute une en-dessous
daemon:*:1:31:The devil himself:/:/sbin/nologin
w
834
Q
$
```

Heureusement on n'a pas à utiliser **ed** tous les jours. Moins on l'utilise, mieux on se porte.

Faute de **ed**, on peut encore lire des fichiers avec **less** ou **more**, voire avec **cat**, et en créer avec cette dernière commande. L'édition devient impossible, sauf de façon automatisée avec des outils comme **sed** et **awk**, que nous verrons plus tard. Voici un exemple d'utilisation de **cat** pour manipuler des fichiers depuis le shell :

```
$ cd /tmp
$ cat > test
Une fois cette commande tapée, tout ce qui est entré va dans le fichier.
Pour terminer le fichier, il faut taper Control-D en début de ligne
^D
$ ls -l test
-rw-r--r--  1 manu  wheel  142 Mar 15 15:32 test
$ cat test
Une fois cette commande tapée, tout ce qui est entré va dans le fichier.
Pour terminer le fichier, il faut taper Control-D en début de ligne
$ cat >> test
Avec cat >> on ajoute à la fin d'un fichier, alors qu'avec cat > on
écrase le contenu précédent.
^D
$ ls -l test
-rw-r--r--  1 manu  wheel  236 Mar 15 15:33 test
$ cat test
Une fois cette commande tapée, tout ce qui est entré va dans le fichier.
Pour terminer le fichier, il faut taper Control-D en début de ligne
Avec cat >> on ajoute à la fin d'un fichier, alors qu'avec cat > on
écrase le contenu précédent.
$
```

Ainsi se finit cette introduction à l'édition des fichiers. Tout cela peut paraître particulièrement atroce, mais il faut garder à l'esprit que ces outils, s'ils sont difficiles à maîtriser, n'en sont pas moins très puissants. L'apprentissage de **vi** sera sans doute la plus grosse difficulté que vous aurez à affronter dans le monde

CULTURE **Caractères de contrôle**

Les caractères de contrôle sont les caractères obtenus en enfonçant la touche **Control** en même temps qu'une autre touche. Ils ont chacun un sens particulier. **Control**+**C** signifie une interruption, **Control**+**[** signifie un échappement, **Control**+**D** signifie fin de fichier... Dans la littérature technique, les caractères de contrôle sont souvent notés ainsi : **^D** pour **Control**+**D**, **^C** pour **Control**+**C**, et ainsi de suite.

PLUS LOIN **Rechercher/remplacer sans vi**

Anticipons un peu l'édition automatisée avec **sed**, avec ici un rechercher/remplacer dans un fichier :

```
# sed 's/rc_configured=NO/rc.configured=YES' /etc/rc.conf > /etc/rc.conf2
# mv /etc/rc.conf2 /etc/rc.conf
```

La commande **sed** sera abordée plus en détails vers la fin de ce chapitre.

Unix (sauf si vous devez un jour configurer **sendmail**), mais une fois qu'on en a acquis la maîtrise, on a du mal à réutiliser le bloc-notes de Windows.

CULTURE **Sendmail**

Sendmail est le serveur de messagerie historique sous Unix. En 2003, il était encore responsable de plus des trois quarts des serveurs de messagerie sur Internet. Il est extrêmement souple, mais sa configuration est hélas très complexe. Courageusement, nous aborderons sa configuration dans le chapitre 12.

Quelques commandes plus avancées pour le shell

Étudions quelques autres commandes utiles du shell.

Variables d'environnement

Sous Unix, tout processus dispose de variables d'environnement. On les manipule comme dans n'importe quel langage de programmation : lire leur valeur, ou la modifier. Exemple en **sh**, **ksh** ou **bash** :

```
$ toto=blablabla
$ echo $toto
blablabla
$ toto="patati patata"
$ echo $toto
patati patata
```

PLUS LOIN **Valeurs avec des espaces**

Si vous voulez donner une valeur comportant des espaces à une variable, il faut ceindre cette valeur d'apostrophes simples (**'**, protection forte) ou doubles (**"**, protection faible).
Pour inclure un caractère spécial, comme une apostrophe simple ou double dans une valeur de variable, on l'« échappe » en le précédant de \ (*antislash*) comme ceci : **pb=l\'incendie**

Le shell transmet ses variables d'environnement aux programmes qu'il démarre. De nombreux comportements en dépendent. Par exemple, la variable `TERM` indique le type du terminal. Un programme comme **vi** l'utilise par exemple pour déterminer le comportement du clavier. Une mauvaise valeur de `TERM`, et les touches fléchées ne fonctionneront plus.

Tout ceci n'est pas très compliqué, à un piège près : les modifications faites aux variables d'environnement ne sont pas visibles aux processus ou programmes invoqués par le shell tant qu'elles n'ont pas été « exportées » avec la commande **export** :

```
$ TERM=vt220
$ export TERM
```

Dans les shells plus récents que le *Bourne Shell*, on peut combiner ces commandes ainsi : **export TERM=vt220**.

La commande **set** (ou **env**) permet de lire la valeur de toutes les variables d'environnement définies. Exemple avec **ksh** :

```
$ set
KSH_VERSION='@(#)PD KSH v5.2.14 99/07/13.2' ❶
LESSCHARSET=latin1 ❷
LINES=24
LOGNAME=manu
MAIL=/var/mail/manu
MAILCHECK=600
MANPATH=/usr/share/man:/usr/pkg/man:/usr/local/man:/usr/X11R6/man ❸
OLDPWD=/home/manu
OPTIND=1
PAGER=less ❹
PATH=/bin:/usr/bin:/usr/pkg/bin:/usr/local/bin:/sbin:/usr/sbin:/usr/pkg/sbin:/us
r/local/sbin: ❺
PPID=264
PS1='$ ' ❻
PS2='> '
PS3='#? '
```

ALTERNATIVE **En csh et tcsh**

En **csh** et **tcsh**, on dispose de commandes différentes selon que l'on veuille définir une variable visible uniquement par le shell ou une variable « exportée ».
Pour une modification locale au shell, on utilise **set** : **set term=vt100**. Pour une variable « exportée », on utilise **setenv** : **setenv TERM vt220**.

```
PS4='+ '
PWD=/home/manu/bsdbook
RANDOM=3018
SECONDS=15572
SHELL=/bin/ksh
TERM=vt102
TMOUT=0
USER=manu
_=set
toto=patati patata ❼
$
```

❶ Variable créée automatiquement par **ksh**, indiquant son numéro de version.

❷ LESSCHARSET indique à la commande **less** le jeu de caractères à utiliser pour l'affichage. La valeur **latin1** permet à **less** d'afficher les accents.

❸ Lors d'une requête d'affichage d'une page **man**, la commande **man** la recherche dans tous les répertoires de la variable MANPATH.

❹ Cette variable est aussi utilisée par la commande **man**. Elle lui indique quelle visionneuse utiliser pour afficher les pages **man**.

❺ PATH joue un rôle similaire à MANPATH : c'est la liste des répertoires où le shell va rechercher toute commande tapée.

❻ PS1 indique l'invite de commande affichée par le shell. Vous pouvez l'enrichir avec un contenu dynamique :

```
$ PS1="--> "
--> PS1="`whoami`@`hostname`$ "
manu@violette$
```

La page **man** du shell utilisé détaillera la syntaxe et les possibilités de cette variable.

❼ La variable toto que nous avons définie dans un exemple précédent, et que le système n'a pas oubliée depuis !

PLUS LOIN **Une invite de commande sophistiquée avec ksh**

En bricolant un peu, on peut obtenir une invite de commande donnant le répertoire courant, sur un **ksh** relativement récent (celui des BSD fait l'affaire) :

```
$ cwd () { cd $@ ; PS1="`whoami`@`hostname -s`<`pwd`> " ; }
$ alias cd='cwd'
$ cd /home/manu
manu@violette</home/manu> cd ..
manu@violette</home> cd /var/spool/mqueue
manu@violette<var/spool/mqueue>
```

Pour configurer automatiquement cette invite ainsi lors de toute nouvelle session, placez la définition de la fonction de shell **cwd** et de l'alias **cd** dans le fichier .profile de votre répertoire personnel. **ksh** exécute les commandes de ce fichier lors de toute connexion utilisateur.

PLUS LOIN PATH **et ./**

Lorsque l'on demande au shell d'exécuter une commande, il la cherche dans tous les répertoires indiqués par la variable PATH.

Pour exécuter un programme situé dans un répertoire absent du PATH, il faudra fournir au shell un chemin menant à la commande, comme par exemple /usr/local/bin/emacs. Si cette commande se trouve dans le répertoire courant, il suffit d'utiliser comme chemin le nom du répertoire courant : **./emacs**.

Signalons une petite différence assez déroutante pour les habitués du DOS : sous DOS, le répertoire courant est toujours utilisé pour chercher des commandes. Pour rétablir ce comportement sous Unix, il suffit d'ajouter le répertoire courant au PATH : **export PATH=$PATH:.** (en **ksh** et **bash**).

Attention toutefois aux implications en matière de sécurité : si vous vous trouvez dans un répertoire où un autre usager a laissé une commande **ms**, faire une faute de frappe (**ms** au lieu de **ls**) vous fera exécuter le programme prévu par l'autre usager. C'est pour cette raison qu'il est déconseillé, et notamment à root, d'avoir . (point) dans son PATH.

Si vous tenez à mettre . dans votre PATH, surtout placez-le à la fin du PATH, sinon une faute de frappe ne sera même plus nécessaire pour vous faire exécuter un faux **ls** !

Gestion des droits des fichiers

On en a parlé brièvement, les droits sur les fichiers sont gérés à travers les droits du propriétaire, du groupe, et du reste du monde. Les détails de la gestion des groupes qui concernent l'administrateur seront abordés au chapitre 6. Voyons comment modifier les droits sur les fichiers.

Trois commandes servent à gérer les droits sur les fichiers : **chmod**, **chown**, et **chgrp**.

chown et **chgrp** servent respectivement à modifier le propriétaire et le groupe d'un fichier. Elles sont assez simples d'emploi et surtout utilisées par root : on indique le nouveau propriétaire ou groupe, suivi de la liste des fichiers affectés.

```
# cd /tmp
# ls -l
total 64
-rw-r--r--  1 root  wheel  29201 Mar 15 12:35 ch3.sgml
drwxr-xr-x  3 root  wheel    512 Apr 16  2001 linux-2.1.14
-rw-r--r--  1 root  wheel    834 Mar 15 15:25 passwd
-rw-r--r--  1 root  wheel    236 Mar 15 15:33 test
# chown manu ch3.sgml test passwd
# chgrp wsrc linux-2.1.14
# ls -l
total 64
-rw-r--r--  1 manu  wheel  29201 Mar 15 12:35 ch3.sgml
drwxr-xr-x  3 root  wsrc     512 Apr 16  2001 linux-2.1.14
-rw-r--r--  1 manu  wheel    834 Mar 15 15:25 passwd
-rw-r--r--  1 manu  wheel    236 Mar 15 15:33 test
```

La commande **chmod** permet de modifier les permissions d'accès d'un fichier. Sa forme la plus simple est d'indiquer

- quels droits modifier : **u** pour utilisateur, **g** pour groupe, et **o** pour les autres (*others*). On peut en préciser plusieurs : **ug**, **og**, ou **uog**.
- la nature de l'opération : **+** pour ajouter des droits, **–** pour en supprimer, et **=** pour fixer une nouvelle valeur indépendamment de la valeur précédente.
- les attributs affectés : **r** pour la lecture (*read*), **w** pour l'écriture (*write*), et **x** pour l'exécution.

Bien entendu, un utilisateur ne peut modifier que les droits des fichiers qu'il possède, sauf root, qui modifie ce qu'il veut. Dans la pratique :

```
$ ls -l ch3.sgml
-rw-r--r--  1 manu  wheel  29201 Mar 15 12:35 ch3.sgml
$ chmod og-r ch3.sgml
$ ls -l ch3.sgml
-rw-------  1 manu  wheel  29201 Mar 15 12:35 ch3.sgml
$ chmod uog+rw ch3.sgml
$ ls -l ch3.sgml
-rw-rw-rw-  1 manu  wheel  29201 Mar 15 12:35 ch3.sgml
$ chmod og=r ch3.sgml
$ ls -l ch3.sgml
-rw-r--r--  1 manu  wheel  29201 Mar 15 12:35 ch3.sgml
```

Ces trois commandes admettent une option **–R** qui permet de modifier récursivement un répertoire et son contenu.

Recherche de fichiers

La commande **find** permet de faire des recherches de fichiers assez complexes. Exemple : recherche des fichiers et répertoires d'extension .so dans /usr/local/jdk

```
$ find /usr/local/jdk -name '*.so' -print
/usr/local/jdk/jre/lib/ppc/green_threads/libhpi.so
/usr/local/jdk/jre/lib/ppc/native_threads/libhpi.so
/usr/local/jdk/jre/lib/ppc/classic/libjvm.so

(...)
```

PLUS LOIN **Notation octale**

Une autre syntaxe est souvent utilisée pour préciser les droits : on indique un nombre à 3 chiffres. Le premier chiffre indique les droits de l'utilisateur, le deuxième ceux du groupe, le troisième ceux des autres. Chaque chiffre est la somme des nombres suivants : 1 pour exécution, 2 pour écriture, et 4 pour la lecture. C'est la notation octale.
Ainsi, taper **chmod 644 test** affecte les droits `-rw-r--r--` au fichier test. **chmod 000 test** affecte les droits `----------`, **chmod 755 test** donne les droits `-rwxr-xr-x`, et ainsi de suite.

PLUS LOIN **Droits par défaut**

Tout fichier nouvellement créé reçoit des droits par défaut, déterminés par un masque. C'est la commande **umask** qui pilote ce masque avec une valeur octale indiquant les droits qui ne doivent pas être attribués. Pour les fichiers créés, le droit par défaut est de 666 moins l'**umask**, et pour les répertoires, c'est 777 moins l'**umask**. Ainsi, un umask de 022 fera créer les fichiers en mode 644 et les répertoires en mode 755.

SUR LES AUTRES UNIX **find**

Sur BSD et GNU/Linux, l'option **-print** n'est pas nécessaire. Sur le **find** de System V, que l'on trouve sur la plupart des Unix commerciaux, le **-print** est obligatoire pour que la commande **find** affiche ses résultats.

La commande **find** a de nombreuses options, qui permettent de spécifier des critères de taille, de propriétaire, de droits, de dates de création ou de modification... Les possibilités sont vastes ; consultez la page **man**.

On veut parfois chercher un fichier sur tout le système, **find / -name 'trucmuche' -print** fait l'affaire, mais cette commande est lente, car les systèmes de fichiers Unix ne sont en général pas conçus pour faire rapidement des recherches de fichiers. Pour pallier ce défaut, les Unix récents disposent de la commande **locate**, qui cherche un fichier dans un index rafraîchi toutes les nuits.

PLUS LOIN **locate**

Sous BSD, l'index se trouve dans /var/db/locate.database. Il est remis à jour chaque nuit, par la commande **locate.updatedb**, située dans /usr/libexec. **locate.updatedb** est invoquée par **cron**, que nous verrons au chapitre 8.

Seul inconvénient de cette méthode : les fichiers ajoutés depuis le dernier rafraîchissement de l'index sont introuvables avec **locate**. Inversement, les fichiers détruits depuis la dernière mise à jour sont toujours présents dans l'index. Exemple d'utilisation :

```
$ locate libhpi.so
/usr/local/jdk/jre/lib/ppc/green_threads/libhpi.so
/usr/local/jdk/jre/lib/ppc/native_threads/libhpi.so
$
```

ATTENTION **locate et son index**

Attention, si l'index de **locate** est vide ou grossièrement obsolète, **locate** n'affiche aucun message d'erreur. Quand **locate** ne trouve pas un fichier, il est donc prudent de vérifier avec **find** avant de conclure.

Enfin, on doit parfois rechercher un contenu précis dans un ensemble de fichiers. C'est le travail de la commande **grep**. Exemple d'utilisation simple :

```
# grep manu /etc/*
exports:/home/manu/xsrc 10.0.12.190
group:wheel:*:0:root,manu
group:operator:*:5:root,manu
group:wsrc:*:9:manu
group:manu:*:500:
netstart:# see the ifconfig manual page for details.
passwd:manu:*:500:500:Emmanuel Dreyfus:/home/manu:/bin/ksh
security:# the mtree(8) manual page.
#
```

CULTURE **Recherche rapides sur Mac**

Le système de fichiers de MacOS (HFS, et son évolution HFS+) est conçu pour faire des recherches très rapides. L'invocation du programme est en général plus longue que la recherche proprement dite.
HFS doit sa célérité à l'utilisation de structures appelées « arbres binaires ».

grep admet l'argument **-r** pour faire des recherches récursives dans un répertoire. Sur certains Unix, cette option n'est pas disponible, et il faut composer une commande un peu plus complexe avec l'option **-exec** de **find** pour invoquer une commande externe sur chaque fichier trouvé (**-type f** indique de limiter la recherche aux fichiers, par opposition aux répertoires, liens et autres).

IMPORTANT **Astérisques et apostrophes**

Le caractère * (astérisque) est souvent utilisé comme joker : il représente n'importe quelle suite de caractères. Lorsque le shell voit un astérisque, il le remplace par la liste des fichiers correspondants. Exemple :

```
$ echo /var/yp/*
/var/yp/Makefile.main
/var/yp/Makefile.yp
/var/yp/binding
$ echo /var/yp/Make*
/var/yp/Makefile.main
/var/yp/Makefile.yp
```

Si l'on tente de passer un astérisque en argument à une commande, il sera intercepté, interprété et remplacé par le shell. Pour empêcher cela, il suffit de le protéger par des apostrophes (**'**). Le shell le laissera alors tel quel.

```
$ find /usr/share/doc -type f -exec grep Multics {} \; -print
The Unix operating system drew many of its ideas from Multics,
/usr/share/doc/smm/05.fastfs/1.t
similar to the scheme used by Multics [Feiertag71] have been added.
/usr/share/doc/smm/05.fastfs/5.t
"The Multics Input-Output System",
/usr/share/doc/smm/05.fastfs/6.t
$
```

EN PRATIQUE **{} \ ; ? au secours !**

La commande ci-contre vous a peut-être donné des sueurs froides. Le bon réflexe pour comprendre ce qu'elle fait est d'aller voir la page **man** de **find**, et plus précisément son option **-exec**. En bref, **-exec** indique à **find** que nous désirons exécuter une commande sur chaque fichier trouvé. Ici, il s'agit de la commande **grep Multics**. **find** remplacera **{}** par le nom du fichier trouvé, et il faut ajouter un **\;** à la fin de la commande pour que **find** s'y retrouve (ce caractère est échappé : ainsi le shell comprend qu'il ne le concerne pas et le passe tel quel à la commande **find**).

La vie sur la machine

On n'est pas toujours seul sur un système Unix. Les commandes **w** et **who** permettent de savoir qui d'autre est connecté à un instant donné, depuis quand, depuis où, et quelle commande il est en train d'exécuter :

```
$ w
 5:03PM  up 30 days,  7:09, 5 users, load averages: 0.16, 0.12, 0.09
USER TTY FROM              LOGIN@  IDLE WHAT
vpn   p0 radium.network.i 25Feb03  18days -ksh
manu  p1 melancolie.net.e 04Mar03  1day -ksh
manu  p2 melancolie.net.e Wed06PM     0 systat
jdoe  p3 s083.dhcp212-198  5:03PM     0 w
manu  p4 melancolie.net.e Wed06PM     0 vi mach_notify.c
$
```

Les commandes **write** et **talk** permettent d'interagir avec un utilisateur : **write** affiche un message sur son terminal et **talk** est utilisé pour initier un dialogue interactif. Voyez les pages **man** de ces commandes pour en savoir plus, ou essayez-les.

EN CAS DE COUP DUR **talk ne fonctionne pas**

La commande **talk** utilise le serveur **talkd**. Si celui-ci n'est pas activé, **talk** ne pourra pas établir la communication.
Si vous êtes administrateur de la machine, vous pouvez activer **talkd** en configurant **inetd**. Nous étudierons la marche à suivre au chapitre 8.

Ça travaille là-dedans : les processus

Unix est un système d'exploitation multi-tâches, c'est-à-dire qu'il permet d'exécuter à la fois plusieurs programmes. Une instance de programme en cours d'exécution s'appelle un processus. Le shell utilisé pour démarrer des commandes sur la machine est un processus. Pour chacune des commandes invoquées, Unix crée un nouveau processus.

La commande **ps** permet d'observer la liste de ses processus en cours d'exécution :

```
$ ps
  PID TT STAT    TIME COMMAND
2155 p0 Ss   0:00.42 -ksh
2182 p0 R+   0:00.15 ps
1337 p1 Ss   0:01.03 -ksh
2166 p1 S+   0:01.13 systat
1296 p3 Ss   0:00.43 -ksh
1311 p3 S+   0:22.87 vi ch3.sgml
$
```

CULTURE **Le PID**

Le PID (*Process ID*) identifie de façon unique chaque processus. C'est un nombre variant en général de 1 à 32767. Certains Unix sont capables d'utiliser des PID plus élevés.
Certains Unix allouent les PID de façon séquentielle, d'autres les choisissent au hasard. Seule règle universelle : deux processus différents ne peuvent pas partager le même PID simultanément.

On trouve ici le PID, le nom du terminal auquel est attaché le processus, son état (voir la page **man** de **ps** pour les détails), le temps qu'il a consommé en exécution sur le processeur, et le nom de la commande invoquée.

Par défaut, **ps** n'affiche que vos processus attachés à un terminal (un dispositif d'affichage sur écran couplé à un dispositif de saisie au clavier). **ps -x** affiche vos processus non attachés à un terminal, et **ps -a** affiche les processus des autres. **ps -ax** affiche tous les processus.

VOCABULAIRE **Tâche de fond**

On parle souvent de processus fonctionnant en « tâche de fond » pour les processus non attachés à un terminal.

```
$ ps -ax
  PID TT STAT    TIME COMMAND
    0 ?? DKs  0:00.09 [swapper]
    1 ?? Ss   0:00.62 init
    2 ?? DK   0:00.00 [scsibus0]
    3 ?? DK   0:00.00 [scsibus1]
    4 ?? DK   0:00.03 [pagedaemon]
    5 ?? DK   0:03.58 [reaper]
    6 ?? DK   0:58.93 [ioflush]
    7 ?? DK   0:00.15 [aiodoned]
   79 ?? Ss   0:00.37 /usr/sbin/syslogd -s
  306 ?? Ss   0:07.09 /usr/sbin/sshd
  315 ?? Is   0:00.14 /usr/sbin/inetd -l
  320 ?? Ss   0:01.73 /usr/sbin/cron
 1293 ?? Ss   0:00.35 sshd: manu [priv]
 1295 ?? S    0:39.57 sshd: manu@ttyp3
 1334 ?? Ss   0:00.38 sshd: manu [priv]
 1336 ?? S    0:01.82 sshd: manu@ttyp1
 2152 ?? Ss   0:00.37 sshd: manu [priv]
 2154 ?? S    0:00.28 sshd: manu@ttyp0
 2155 p0 Ss   0:00.44 -ksh
 2186 p0 R+   0:00.10 ps -ax
 1337 p1 Ss   0:01.03 -ksh
 2166 p1 S+   0:01.55 systat
 1296 p3 Ss   0:00.43 -ksh
 1311 p3 S+   0:25.17 vi ch3.sgml
  322 E0 S    0:00.56 /usr/libexec/getty std.9600 ttyE0
  323 00 Is+  0:00.20 /usr/libexec/getty std.38400 tty00
$
```

Cette liste est issue d'un NetBSD 1.6. On y trouve un certain nombre de processus fonctionnant en mode noyau, dont les noms sont entre crochets. Ne vous souciez pas trop de ces processus : vous ne pourrez ni les tuer ni les redémarrer, car ils font partie intégrante du système.

On trouve également quelques programmes en tâche de fond, qui assurent des services : **syslogd**, **cron**, **inetd**, **sshd**, deux **getty** qui attendent une identification de session sur leurs terminaux d'attache, et les commandes démarrées par un utilisateur qui est justement en train de taper le présent chapitre dans **vi**.

ps peut afficher de nombreuses autres informations pour chaque processus. Pour en savoir plus, voyez sa page **man**.

La commande **kill** permet de tuer un processus. Pour tuer le processus **systat**, il suffit d'invoquer **kill** en indiquant son PID : **kill 2166**

kill envoie un signal au processus lui demandant de se terminer proprement. Si le processus est vraiment planté, cela peut ne pas suffire, il faut alors requérir sa destruction immédiate et sans délais, avec le signal **-9** : **kill -9 2166**

SUR LES AUTRES UNIX **Des divergences de ps**

La syntaxe de la commande **ps** diffère malheureusement fortement entre BSD et GNU/Linux d'une part et System V d'autre part. Sur les UNIX System V, il faut taper **ps -ef** pour avoir tous les processus.
Les anciens systèmes BSD ont aussi une syntaxe particulière, sans **-** : il faut taper **ps ax** au lieu de **ps -ax**.
Pour compliquer le tout, il y a des hybrides : AIX d'IBM, qui est un System V, utilise la syntaxe BSD, et certains Unix comprennent les deux syntaxes, en faisant la distinction sur la présence ou non du **-**.

B.A.-BA **Le noyau**

Le noyau (*kernel* en anglais) est le cœur du système d'exploitation. C'est lui qui suspend et qui reprend l'exécution des processus des utilisateurs à intervalles de temps régulier, donnant ainsi l'impression d'un système multi-tâches. Il doit aussi fournir aux processus l'accès au matériel et à diverses ressources telles que le système de fichiers ou les terminaux. C'est encore lui qui assure la sécurité du système, en refusant par exemple de laisser un utilisateur modifier un fichier sur lequel il n'a pas de droits.

PLUS LOIN **Processus morts vivants**

Un processus tué peut ne pas disparaître immédiatement si le système a encore besoin de lui pour référence. Le processus apparaît alors toujours sous **ps**, avec un nom entre parenthèses, et un `Z` dans la colonne d'état : c'est un processus zombie. Tant que les zombies apparaissent de façon isolée, ne vous en inquiétez pas : ils ne consomment pas de ressources, et disparaîtront d'eux-mêmes lorsqu'ils auront réglé les affaires de famille qui les retiennent dans le monde des vivants.
Un grand nombre de zombies indique par contre un dysfonctionnement d'une commande. Si votre système est peuplé de zombies, il est sans doute temps de faire un rapport de bogue.

Devenez un grand sorcier avec le shell

Cette section donne des ouvertures vers des utilisations plus avancées du shell. Il est bien entendu inutile de tout maîtriser immédiatement, vous aurez tout le temps de vous familiariser avec tout cela à mesure que vous utiliserez Unix.

Un peu de plomberie

Une des grandes forces d'Unix, c'est sa capacité de renvoyer la sortie d'une commande à l'entrée d'une autre. Cela se fait dans le shell avec les tuyaux (*pipes* en anglais). L'utilisation la plus simple consiste à invoquer une commande qui affiche plus d'une page de résultats et à envoyer sa sortie dans **more** ou **less** pour pouvoir tout examiner convenablement : **find / -type f | less**

On peut donc emboîter des tuyaux de commandes pour faire travailler des commandes à la chaîne. Exemple détaillé :

```
$ ps -ax | grep systat
2198 p0 S+   0:00.13 grep systat
2166 p1 S+   0:02.64 systat
$ ps -ax | grep systat | awk '{print $1}'
2201
2166
$ ps -ax | grep systat | awk '{print $1}' | xargs kill
kill: 2203: No such process
$
```

Le processus **grep**, de PID 2203, est déjà terminé au moment de l'exécution du **kill**, d'où le message d'erreur. En revanche, le processus **systat**, de PID 2166, n'en réchappera pas.

Sortie standard et erreur standard

Chaque commande produit deux flux : la sortie standard et l'erreur standard, où sont évidemment envoyés les messages d'erreur. Ces deux flux s'affichent à l'écran en temps normal. Les commandes disposent également d'un flux d'entrée standard.

Lorsque l'on utilise des tuyaux, seule la sortie standard est transmise dans un tuyau au processus suivant, sur son entrée standard. L'erreur standard va continuer à s'afficher à l'écran. Pour éviter cela et la joindre à la sortie standard, il faut faire une redirection comme ceci (syntaxe valable en **sh**, **ksh** et **bash**) :

```
$ find /etc -type f -exec grep truc {} \;  2>&1 | less
# reports of network infrastructure difficulties
grep: /etc/hosts.equiv: Permission denied
grep: /etc/master.passwd: Permission denied

(...)
```

PLUS LOIN **man, less, et les tuyaux**

Les tuyaux, c'est exactement ce que la commande **man** utilise pour afficher ses pages. On peut la simuler ainsi : **groff -Tascii -man /usr/share/man/man1/ls.1 | less**
La ligne ci-dessus est très utile lorsque l'on veut lire une page **man** située à un endroit où la commande **man** ne va pas la chercher…

ALTERNATIVE **En csh et tcsh**

En **csh** et **tcsh**, la redirection dans un tuyau se fait ainsi : **find /etc -type f -exec grep truc {} \; |& less**
Et comme suit dans un fichier : **find /etc -type f -exec grep truc {} \; >& logerr**

On peut aussi rediriger la sortie standard ou l'erreur standard vers un fichier pour consultation ultérieure, ou vers /dev/null pour s'en débarrasser (syntaxe valable en **sh**, **ksh** et **bash**) :

```
$ grep -r truc /etc 2> err > out
$ ls -l err out
-rw-r--r--  1 manu  wheel  629 Mar 15 17:25 err
-rw-r--r--  1 manu  wheel  220 Mar 15 17:25 out
$ cat err
grep: /etc/hosts.equiv: Permission denied
grep: /etc/master.passwd: Permission denied
grep: /etc/skeykeys: Permission denied
grep: /etc/spwd.db: Permission denied
grep: /etc/ppp/pap-secrets: Permission denied
grep: /etc/ppp/chap-secrets: Permission denied
grep: /etc/ssh_host_key: Permission denied
grep: /etc/ssh_random_seed: Permission denied
grep: /etc/X11/xdm/authdir: No such file or directory
grep: /etc/ssh_host_dsa_key: Permission denied
grep: /etc/ssh/ssh_host_key: Permission denied
grep: /etc/ssh/ssh_host_dsa_key: Permission denied
grep: /etc/ssh/ssh_host_rsa_key: Permission denied
grep: /etc/cgd: Permission denied
$ cat out
/etc/aliases:# reports of network infrastructure difficulties
/etc/mail/aliases:# reports of network infrastructure difficulties
/etc/rc.d/raidframe:    # Initiate parity/mirror reconstruction as needed, in th
e background.
$ grep -r truc /etc 2>/dev/null > out2
$ ls -l out2
-rw-r--r--  1 manu  wheel  220 Mar 15 17:26 out2
$ cat out2
/etc/aliases:# reports of network infrastructure difficulties
/etc/mail/aliases:# reports of network infrastructure difficulties
/etc/rc.d/raidframe:    # Initiate parity/mirror reconstruction as needed, in th
e background.
$
```

PLUS LOIN **/dev/null**

/dev/null est un fichier spécial qui n'a pas de fond : une sorte de trou noir où tout ce qui est écrit est perdu, sans occuper de place sur le disque.

Manipulation de texte avec sed et awk

Quelques commandes, telles que **grep**, **tr**, **sed**, ou **awk**, permettent des manipulations de texte extrêmement puissantes. Nous avons vu que **grep** servait à rechercher des chaîne de caractères dans les fichiers. Voyons rapidement les usages les plus courants de **sed** et **awk**.

sed sert à travailler sur un fichier ligne à ligne. Son usage le plus courant est le rechercher/remplacer, que l'on peut limiter à certaines lignes, comme dans **vi** :

```
$ cat > /tmp/test
Ceci est un test
foo bar buz bar
bidon
^D
$ sed 's/bar/foo/g' /tmp/test
Ceci est un test
foo foo buz foo
bidon
$ sed '2,$s/u/U/' /tmp/test
Ceci est un test
foo bar bUz bar
bidon
$ sed 's/bar$/BAAAR/' /tmp/test > /tmp/test2
$ cat /tmp/test2
Ceci est un test
foo bar buz BAAAR
bidon
$ sed 's/foo \(.\{1,3\}\) buz/\1/' /tmp/test2
Ceci est un test
bar BAAAR
bidon
$
```

PLUS LOIN **Expressions régulières**

Les derniers exemples de **sed** mettent en jeu des expressions régulières. **bar$**, signifie que l'on veut un **bar** ancré en fin de ligne. De même, on pourrait indiquer **^foo** pour un **foo** ancré en début de ligne.

Le bouquet final montre un usage plus complexe d'expression régulière. **.** (point) signifie n'importe quel caractère, **\{1,3\}** l'atome précédent (n'importe quel caractère) de 1 à 3 exemplaires, et **sed** remplace le **\1** dans la partie « remplacer » par ce qu'il a trouvé dans la première paire de **\(\)**. Tout ceci est également utilisable dans la commande rechercher/remplacer de **vi**.

sed fait d'autres choses que des rechercher/remplacer, mais nous allons nous en tenir à cela pour le moment. Vous vous rendrez rapidement compte que **sed** est assez handicapé pour gérer les retours chariot. **tr** est mieux adapté à cet usage :

```
$ cat /tmp/test | tr '\n' ' '
Ceci est un test foo bar buz bar bidon
$
```

awk est plus utilisé pour travailler sur des colonnes. On lui donne des commandes à appliquer à toutes les lignes d'un fichier, où **$1** représente la première colonne, **$2** la deuxième, etc. On peut indiquer le séparateur de colonnes, des conditions sur les lignes à traiter, des instructions à exécuter avant ou après traitement du fichier. Les possibilités sont très vastes : on peut même faire des boucles `while` ou des tests `if` avec la même syntaxe qu'en C.

Quelques exemples simples :

```
$ ps
  PID TT STAT    TIME COMMAND
2155 p0 Ss   0:01.07 -ksh
2336 p0 R+   0:00.10 ps
1337 p1 Ss+  0:01.03 -ksh
1296 p3 Ss   0:00.44 -ksh
1311 p3 S+   0:36.96 vim ch3.sgml
$ ps |awk '{print $2}'
TT
p0
p0
p0
p1
p3
p3
$ ps |awk '{if (NR != 1){print $2}}'
p0
p0
p0
p1
p3
p3
$ ps |awk '{if (NR != 1){print $2}}' | sort -u
p0
p1
p3
$ awk -F: '{if ($4 == 0){print $1}}' /etc/passwd
root
toor
$ awk 'BEGIN {FS=":"; ORS=","} {if ($4 == 0){print $1}}' /etc/passwd
root,toor,
$ awk 'BEGIN {FS=":"; ORS=","} {if ($4 == 0){print $1}}' /etc/passwd | sed 's/,$//'
root,toor
$
```

PLUS LOIN **Traductions des retours chariot**

Les systèmes d'exploitation les plus répandus sont Windows, MacOS et Unix. À eux trois, il ont réussi l'exploit d'adopter trois conventions différentes pour les retours chariot dans les fichiers texte.
Windows utilise *Carriage Return* (noté `CR`, `^M`, ou `\r`) suivi de *Line Feed* (noté `LF`, `^J`, ou `\n`). MacOS utilise *Carriage Return* seul, et Unix utilise *Line Feed* seul.

tr peut être utilisé pour visualiser sous Unix un fichier texte venant de MacOS : **`tr '\r' '\n' < fichiermac.txt`** (cette syntaxe, valable en **sh**, **ksh** et **bash**, redirige l'entrée standard de la commande depuis le fichier indiqué).

Montez votre machine infernale : les scripts shell

Un script shell est une liste de commandes de shell à exécuter. Pour faire un script shell, c'est très simple, il suffit d'une ligne magique au début du fichier, qui indique le chemin de l'interpréteur à invoquer sur le script, précédé par les caractères **#!** (dièse point d'exclamation).

```
$ cat > /tmp/script.sh
#!/bin/sh

echo "Hello world"
^D
$ chmod uog+rx /tmp/script.sh
$ /tmp/script.sh
Hello world
```

Nous pouvons finir cette section par quelques constructions de shell que vous pourrez revenir piocher le jour où vous aurez besoin de faire un script. Ce petit script ne prétend être exhaustif : il se contente d'illustrer quelques exemples de programmation en shell.

```
#!/bin/sh

# Attribution de variable: pas d'espaces autour du =
# pas de dollar devant le nom de la variable
hello="Hello world"

# La valeur d'une variable s'obtient en lui préfixant un $
echo $hello

# Tests sur des chaînes: il faut ajouter un caractère devant
# la variable pour éviter des problèmes si elle est vide.
# Les espaces autour des crochets sont obligatoires.
# Voir la page man test(1) pour les tests disponibles.
if [ "x$hello" = "xHello world" ] ; then echo "yes"; fi

# Les shells anciens ne connaissent pas le «non» logique. Il faut le simuler
# avec un «ou» logique (la seconde partie du «ou» n'étant évaluée que si
# la première partie est fausse, le «ou» a l'effet d'un «si ce qui suit est
# faux, exécute la deuxième partie»)
test -x /etc/passwd || echo "/etc/passwd n'est pas exécutable"

# De même, on peut conditionner l'exécution d'une commande à une autre
# à l'aide d'un «et» logique: si ce qui suit est vrai, exécute la suite
grep manu /etc/passwd > /dev/null && echo "J'ai un compte ici"

# On peut donner à une variable le résultat d'une commande avec $( )
login=$( awk -F: '{if ($4 == 500){print $1}}' /etc/passwd )

# On peut annuler l'effet d'un caractère spécial (ex: $) avec \
# Le shell interprète le contenu dans "" mais pas dans ''
var=1
echo "\$var vaut $var"   # affiche $var vaut 1
echo '\$var vaut $var'   # affiche \$var vaut $var

# Les compteurs se font avec la commande expr.
# Les espaces autour des crochets sont toujours obligatoires.
# Les `` sont un synonyme de $( ), mais on ne peut pas les emboîter.
i=1
while [ $i -lt 10 ] ; do
        echo "i = $i"
        i=`expr $i + 1`
done
```

EN CAS DE COUP DUR Le script shell ne fonctionne pas

Un script shell peut ne pas fonctionner pour différentes raisons : oubli des droits en exécution, ou la ligne **#!** n'introduit pas un chemin d'exécutable existant.
Parfois, le script démarre correctement mais ne fonctionne pas comme on le voudrait. L'option **-x** de **sh** permet d'afficher exactement tout ce qui se passe :

```
$ sh -x /tmp/script.sh
+ echo Hello world
Hello world
```

PLUS LOIN Les scripts Perl

Le shell a l'avantage d'une compatibilité assez large, mais il est assez inconfortable à l'usage. Beaucoup d'administrateurs préfèrent écrire leurs scripts en Perl, un langage interprété taillé pour cela.
Perl est inclus dans la distribution de base d'OpenBSD et de FreeBSD avant la version 5.0. Il est fourni dans le système de paquetages pour NetBSD et FreeBSD 5.0. Pour vous mettre le pied à l'étrier, voici un script **perl** simple (attention, la première ligne doit indiquer l'emplacement de l'exécutable **perl** sur votre système) :

```
#!/usr/bin/perl
print "hello world\n";
```

Perl est un langage aux possibilités très vastes. Pour l'apprendre, un livre sera probablement le bienvenu, mais à défaut vous pouvez commencer par sa page **man** : `perl(1)`.

D'une machine à l'autre

Si vous disposez de comptes sur plusieurs machines UNIX différentes, vous ne tarderez pas à avoir besoin de vous connecter d'une machine à l'autre – pour invoquer des commandes ou copier des fichiers.

L'ancien monde : `telnet`, FTP, `rlogin`, `rsh`, et `rcp`

Sur les UNIX les plus anciens, les services **`telnet`** et **`rlogin`** étaient disponibles pour démarrer une session à distance, et le service **`rsh`** était souvent utilisé pour exécuter une unique commande sur une machine distante. Pour les copies de fichier, que ce soit depuis ou vers la machine proposant le service, FTP et **`rcp`** ont un temps été les standards. Ces protocoles s'utilisaient respectivement avec les commandes **`telnet`**, **`rlogin`**, **`rsh`**, **`ftp`**, et **`rcp`**. Exemple :

```
plume$ telnet spoutnik.example.net
Trying 192.0.2.15...
Connected to spoutnik.example.net
Escape character is '^]'.


UNIX(r) System V Release 4.0 (spoutnik)

login: manu
password: azerty
spoutnik$ exit
Connection closed by foreign host
plume$ ftp spoutnik.example.net
Connected to spoutnik.example.net.
220-
220 192.0.2.15 FTP server ready.
Name (spoutnik.example.net:manu): manu
331 Password required for manu.
Password: azerty
230 User manu logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> get temp
local: temp remote: temp
229 Entering Extended Passive Mode (|||15861|)
150 Opening BINARY mode data connection for 'temp' (281500 bytes).
281500 bytes received in 00:06 (44.88 KB/s)
ftp> put group
local: group remote: group
229 Entering Extended Passive Mode (|||17913|)
150 Opening BINARY mode data connection for 'group'.
226 Transfer complete.
317 bytes sent in 00:00 (1.44 KB/s)
ftp> quit
plume$
```

B.A.-BA **Particularités de FTP**

FTP est le seul rescapé de tous ces services, puisqu'il reste couramment utilisé sur Internet comme protocole de mise à disposition de fichiers pour le public. On parle alors de FTP anonyme : la connexion se fait en tant que l'utilisateur **`ftp`** ou **`anonymous`**, sans mot de passe – certains sites exigent qu'on tape à la place son adresse électronique. Dans tous les cas, aucun vrai mot de passe ne transitant, le fait que FTP ne chiffre rien n'est pas un problème.

FTP dispose de fonctionnalités utiles, comme la reprise d'un téléchargement interrompu (commande **`reget`** de **`ftp`**). À l'inverse, la fonctionnalité de transfert de fichiers en mode ASCII ou binaire est une éternelle source de corruption de fichiers pour les usagers qui n'y prennent pas garde : si vous devez transférer des fichiers qui ne contiennent pas que du texte brut, assurez-vous de bien être en mode binaire (commande **`binary`** de **`ftp`**), que vous préférerez au moindre doute.

ATTENTION **Invites de commande**

Dans ces exemples, on travaille sur plusieurs machines différentes. Pour éviter les confusions, l'invite de commande `$` est ici préfixée du nom de la machine à laquelle on s'adresse. Bien entendu, ce nom fait partie de l'invite de commande et il ne faut pas le taper.

Exploitation à distance avec SSH

Tous ces services étaient fondamentalement non sûrs dans la mesure où ils laissaient circuler les mots de passe en clair à travers le réseau, ce qui facilitait la tâche aux pirates. Les services **rlogin**, **rsh** et **rcp** disposaient de plus de mécanismes permettant de passer d'une machine à l'autre sans avoir à donner de mot de passe, véritable catastrophes sur le plan de la sécurité. Ils sont donc tombés en désuétude et ont été remplacés par un unique service : SSH. Ce dernier utilise une cryptographie forte pour assurer des communications authentifiées, confidentielles, et inaltérées.

ATTENTION **Pas de communication sans serveur**

Que ce soit pour les commandes **telnet**, **ftp**, **rlogin**, **rsh**, **rcp**, ou pour les plus récentes **ssh**, **scp** et **sftp**, aucune commande n'est utile si la machine cible n'offre pas le service correspondant. Ainsi, si la machine sur laquelle vous désirez vous connecter n'a pas de serveur SSH en fonction, vous ne pourrez pas utiliser **ssh**, **scp** ou **sftp**.

La commande **ssh** permet de se connecter à une machine proposant le service SSH pour entamer une session, à la manière de **telnet** ou **rlogin**, ou pour exécuter une simple commande, à la manière de **rsh**. Les commandes **scp** et **sftp** permettent les échanges de fichiers avec un serveur SSH, et leurs interfaces ressemblent respectivement à celles de **rcp** et **ftp**. Voici quelques exemples d'utilisation :

```
plume$ ssh violette
manu@violette's password: azerty
violette$ exit
plume$ ssh violette ls -l /dev/wd0a
manu@violette's password: azerty
brw-r-----  1 root  operator  13, 0 Jul  7  2003 /dev/wd0a
plume$ ssh guest@violette
guest@violette's password: guest
violette$ exit
plume$ scp violette:/etc/passwd ./
manu@violette's password: azerty
passwd                                        100%  830    42.1KB/s   00:00
plume$ ls -l passwd
-rw-r--r--  1 manu  manu  830 Jan 17 15:57 passwd
plume$ scp /etc/group violette:/tmp
manu@violette's password: azerty
group                                         100%  317    98.8KB/s   00:00
plume$ ssh violette ls -l /tmp/group
manu@violette's password: azerty
-rw-r--r--  1 manu  wheel  317 Jan 17 15:58 /tmp/group
plume$ sftp violette
Connecting to violette...
manu@violette's password: azerty
sftp> cd /tmp
sftp> ls
.
..
.font-unix
group
sftp> get group
group                                         100%  317     1.4KB/s   00:00
sftp> quit
plume$ ls -l group
-rw-r--r--  1 manu  manu  317 Jan 17 16:00 group
plume$
```

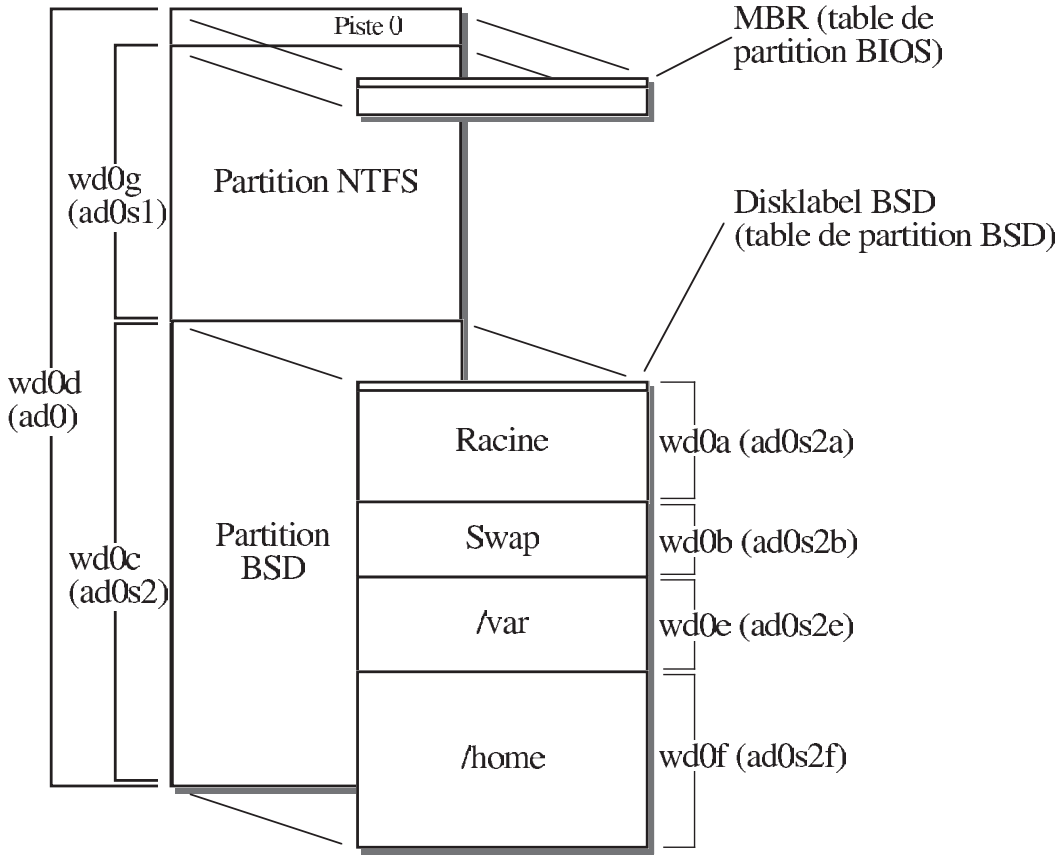
ASTUCE **Quitter une session SSH**

Il arrive parfois qu'une session SSH reste bloquée, quand par exemple le shell sur la machine distante s'est planté. On peut alors forcer la clôture de la session SSH en tapant la séquence d'échappement suivante : **~.** (tilde suivi de point) – ceci nous amène à une autre astuce : comme il sert aux séquences d'échappement, on ne peut pas afficher un tilde directement dans une session SSH. Pour l'obtenir, il faut taper **~~** (tilde deux fois).
La séquence d'échappement pour fermer une session **telnet** est différente : il faut taper **Control+]** puis entrer la commande **quit**.

ATTENTION **Mauvais mot de passe**

Le mot de passe employé dans ces exemples, **`azerty`**, est un très mauvais mot de passe, beaucoup trop simple à deviner. Pour un bon niveau de sécurité, vous devez choisir des mots de passe complexes, comportant des caractères variés (majuscules, minuscules, caractères de ponctuation) et n'étant un mot ou « presque » un mot dans aucune langue (« ball0n » est ainsi déconseillé).
Une méthode consiste à adapter les initiales d'une phrase. « Je vous prie d'agréer, Monsieur » deviendrait ainsi « Jvpd'a,M ».

4


Piste 0
MBR (table de partition BIOS)
wd0g (ad0s1)
Partition NTFS
Disklabel BSD (table de partition BSD)
wd0d (ad0)
Racine
wd0a (ad0s2a)
Partition BSD
Swap
wd0b (ad0s2b)
wd0c (ad0s2)
/var
wd0e (ad0s2e)
/home
wd0f (ad0s2f)

Installer un système BSD

Ce chapitre survolera les étapes de l'installation d'un système BSD. Le terme « survol » n'est pas employé à la légère : les procédures d'installation varient un peu d'une version à l'autre, beaucoup selon le système installé, énormément en fonction de l'architecture. Pour éviter de se perdre dans une abondance de détails vouée à une obsolescence rapide, on se limitera donc à une vision plutôt globale des différentes étapes.

SOMMAIRE

MOTS-CLEFS

- Partitionnement
- Installation

EN PRATIQUE **Sites Web des BSD**

- http://www.netbsd.org
- http://www.freebsd.org
- http://www.openbsd.org

Où commencer ?

C'est la première question à se poser. Pour tous les BSD, la réponse est la même : par le site Web du projet. NetBSD, FreeBSD, et OpenBSD y présentent tous les matériels pris en charge, la manière de procéder, et où demander de l'aide si les choses tournent mal.

Ces sites Web regorgent d'informations utiles : la documentation, les archives des listes de diffusion du projet, les systèmes de rapport de problèmes, et les actualités. C'est aussi l'endroit d'où télécharger le système lui-même. Il faut donc vraiment prendre l'habitude de les consulter.

Problèmes de prise en charge du matériel

Commençons par la liste du matériel pris en charge. On trouve cette information assez rapidement sur la page d'accueil de chacun des projets (moyennant de savoir que matériel se dit « *hardware* » en anglais). Pour chacun des trois BSD, il faut indiquer la plateforme dont on veut la liste du matériel reconnu. Dans le cas de NetBSD, une bonne partie des cartes sont gérées par des pilotes indépendants des plateformes, listés séparément.

Tout matériel absent de ces listes ne doit pas forcément être écarté : il se peut que la liste ne soit pas parfaitement à jour ou qu'une version de développement du noyau soit capable de le traiter. Pour s'en assurer, quelques heures d'exploration du site Web du projet seront nécessaires. Si les Foires Aux Questions (FAQ), archives de listes de diffusion, et bases de données de matériel reconnu restent muettes à ce sujet, il est toujours possible de tenter un message sur une liste de diffusion. Attention : assurez-vous auparavant que l'information est introuvable, et choisissez bien la liste. Sans ces efforts minimaux, votre question risque d'être traitée par le mépris.

PLUS LOIN **BSD-current**

Chacun des BSD dispose d'une version de développement, appelée *-current*, sur laquelle travaillent les programmeurs. Régulièrement, une version stable (dite de « *release* ») sort, destinée à une utilisation en production. La version de développement incorpore les toutes dernières nouveautés, mais n'a pas été autant éprouvée que les versions stables. Elle n'est malgré tout pas forcément un cocktail explosif, et certains usagers particulièrement confiants l'utilisent au jour le jour sur des serveurs en production. Mais il faut être conscient du fait qu'il est nettement plus probable d'avoir une mauvaise surprise avec *-current* qu'avec une version stable.

Tout problème de matériel non reconnu se résoudra de l'une des manières suivantes : utilisation de la version *-current* du système, capable de gérer la carte exotique ; achat d'une autre carte mieux prise en charge ; choix d'un autre système BSD ; ou abandon de l'expérience BSD, car après tout il y a d'autres belles choses à essayer dans la vie, alors pourquoi perdre son temps sur des histoires de cartes mal gérées ?

À tous ceux qui n'ont pas opté pour la dernière solution, nous pouvons maintenant présenter l'installation proprement dite.

Les notes d'installation

Un seul conseil : lisez les notes d'installation. C'est la seule documentation à jour qui permette d'éviter tout écueil. Sur OpenBSD et FreeBSD, les notes d'installation sont les mêmes pour toutes les architectures. Au moment où sont écrites ces lignes,

seules les plateformes les plus populaires pour ces systèmes sont abordées dans les notes : i386 et sparc pour OpenBSD, et i386 et Alpha pour FreeBSD. Pour les autres plateformes, il faut aller chercher d'éventuelles informations sur les pages Web correspondantes, voire se débrouiller seul.

Pour NetBSD, les notes d'installation sont spécifiques à chaque plateforme proposée : il faut donc d'abord consulter la page de la plateforme ciblée. Signalons que comme pour FreeBSD et OpenBSD, on trouve des ports plus expérimentaux du système, où personne n'a encore écrit de notes d'installation. Il ne faut pas en conclure pour autant que le port est immature et que vous vous apprêtez à explorer des territoires vierges. Dans une telle situation, il peut être bon de consulter les archives de la liste de diffusion du port, voire d'y demander de l'aide. Si de plus vous arrivez à vos fins, songez à écrire la documentation, à l'intention de la prochaine personne qui s'y risquera.

Les notes d'installation de NetBSD diffèrent aussi de celles des autres BSD par la présence d'une assez longue introduction présentant NetBSD, les nouveautés de la dernière version, et toutes les raisons pour lesquelles le choix de NetBSD s'avère judicieux. Les notes d'installation en elles-mêmes se trouvent un peu plus bas dans le document.

VOCABULAIRE **Architectures**

Les noms des architectures peuvent sembler un peu opaques au non initié. Quelques exemples :

- i386 désigne les PC et compatibles. Le nom vient du 80386 d'Intel, le plus ancien processeur de la famille 80x86 capable d'accueillir Unix. Ce terme recouvre les processeurs suivants : 80486, Pentium (c'est un 80586), et Pentium Pro, II, III, ou IV (ce sont des 80686).
- mac68k désigne les Macintosh à base de processeur 680x0. Le *k* est celui de « kilo » : 68k est donc un raccourci pour 68000. macppc désigne les Macintosh à base de processeur PowerPC (FreeBSD les appelle « ppc »).
- sun2, sun3, sparc et sparc64 désignent les différentes générations de stations Sun, utilisant les processeurs 68010, 68020 ou 68030, Sparc (stations sun4, sun4c and sun4m), et UltraSparc (stations sun4u).
- La liste continue... Pour FreeBSD et OpenBSD, le tour de la question est vite fait. Dans le cas de NetBSD, on compte plus de cinquante architectures différentes.

Installation classique par le programme d'installation

L'installation de tout système d'exploitation se déroule comme suit :

- partitionnement du disque ;
- formatage des partitions ;
- installation des fichiers composant le système ;
- installation de l'amorce du disque ;
- configuration du minimum vital pour que le système puisse démarrer.

Pour les trois systèmes BSD, ces opérations se mènent en général en démarrant un noyau d'installation. Ce dernier contient dans un disque virtuel un système de fichiers où l'on trouve les différents outils utiles au partitionnement, au formatage, à l'installation des fichiers, à l'installation de l'amorçage, et à une configuration minimale. Les fichiers à installer peuvent provenir d'un CD-ROM ou d'un serveur FTP ; le programme d'installation est capable de les chercher là où ils se trouvent.

Selon le matériel utilisé, on peut démarrer le noyau d'installation en utilisant une disquette (c'est souvent la méthode la plus simple), un CD-ROM, à travers le réseau, voire depuis une bande magnétique.

VOCABULAIRE **Ports**

En terminologie NetBSD, un « port » est une version de NetBSD fonctionnant sur une plateforme. Le port est le résultat du portage de NetBSD sur cette plateforme.
Pour FreeBSD et OpenBSD, un port est un paquetage source. Nous aborderons ces sujets au chapitre 11.

VOCABULAIRE **Le « bootstrap »**

L'amorce du système d'exploitation se traduit en anglais par le terme « *bootstrap* ». Il s'agit d'un petit programme chargé d'invoquer le noyau du système d'exploitation lors du démarrage.

PLUS LOIN **Le « netboot »**

L'exception notable à cette démarche est le cas des installations sur des machines sans disque dur, qui doivent démarrer à travers le réseau. On parle alors de « *netboot* ».

Résolution des conflits

Le démarrage du noyau d'installation est particulier dans les versions de FreeBSD antérieures à la version 5.0 : on se voit offrir la possibilité d'activer ou de désactiver des pilotes de matériel. Cela est dû au fait que FreeBSD reconnaît un grand

nombre de cartes ISA non PNP, dont le sondage en aveugle peut planter la machine. Il est donc demandé à l'usager de renseigner un minimum sa configuration.

Les cartes ISA récentes sont PNP, et le bus ISA est de toutes façons en passe de disparaître des PC modernes, Le projet FreeBSD a donc décidé de rendre l'étape de résolution de conflits facultative à partir de FreeBSD 5.0.

NetBSD et OpenBSD ont adopté une fonctionnalité appelée **`userconf`**, similaire à l'étape de résolution de conflits de FreeBSD. Intégrée récemment, cette étape a toujours été facultative, pour la même raison : les cartes ISA non PNP se font désormais rares.

Choix du disque

Une fois le démarrage terminé, le programme d'installation prend la main. Il commence par demander sur quel disque le système doit être installé. Si la machine ne compte qu'un disque, le choix est vite fait, mais il y a un risque de se tromper dans le cas contraire. Si les disques sont de tailles différentes, on aura une chance de s'apercevoir de son erreur au moment de définir la table de partitions, avant d'avoir effectivement partitionné. Mais si les disques sont identiques et si certains d'entre eux contiennent des données que l'on souhaite préserver, la solution la plus simple consiste à les débrancher. Elle a pour inconvénient de forcer à un réajustement une fois que tous les disques seront remis en fonction, les noms des disques sous BSD étant susceptibles de changer lors de toute variation de leur configuration dans la machine.

PLUS LOIN **Fixer le nom des disques**

Il est possible sous FreeBSD de fixer les noms des disques et ainsi éviter les éventuels changements de noms automatiques à toute suppression ou ajout de disque. Cela se fait dans le fichier de configuration du noyau (que nous aborderons au chapitre 13) pour les versions antérieures à FreeBSD 5.0 et dans le fichier /boot/device. hints à partir de FreeBSD 5.0. Consultez la documentation pour plus d'informations.

Cohabitation avec un autre système d'exploitation

Étape suivante : le partitionnement du disque. Un système BSD a besoin d'au moins deux partitions : pour la racine du système de fichiers, et pour la mémoire virtuelle, de pagination (*swap*). On ajoute souvent d'autres partitions pour /var, /usr, ou d'autres arborescences, mais cela n'est pas obligatoire.

Les systèmes BSD utilisent tous une même structure pour stocker les informations de partitionnement. Appelée *disklabel*, elle est enregistrée vers le début du disque

CULTURE **Le bus ISA**

Le bus ISA est l'ancien bus d'extension des compatibles PC, remplacé dans les machines modernes par les bus PCI et AGP. Il a pour seul avantage qu'il est simple pour un électronicien amateur de faire des cartes ISA, mais il souffre d'un inconvénient majeur : il n'existe pas de mécanisme pour détecter de manière sûre quelle carte est installée dans un emplacement (« *slot* ») ISA. Le système d'exploitation peut donc tenter de sonder le matériel, ou demander à l'utilisateur la configuration des cartes ISA présentes : numéro d'interruption (IRQ) et adresse du port d'entrée/sortie.
Les sondages se font en écrivant et en lisant dans des registres de la carte, en supposant être en présence d'une carte particulière à un numéro d'interruption (IRQ) et à une adresse d'entrée/sortie donnés. Le problème, c'est qu'un sondage destiné à une carte *X* mais touchant une carte *Y* peut perturber cette dernière, et avoir une issue fatale. Comme la détection automatique ne peut pas fonctionner dans tous les cas de figure, FreeBSD demandait à l'utilisateur de résoudre ces conflits au démarrage.
Le standard PNP (*Plug'N Play*, les mauvaises langues disent *Plug'N Pray*) permet à la machine de sonder et de configurer les cartes ISA sans risque. Il ne fonctionne évidemment qu'avec les cartes ISA PNP.

(l'emplacement exact dépend des plateformes). L'opération de partitionnement installe donc un *disklabel* décrivant les partitions souhaitées.

Installation dédiée ou cohabitation ?

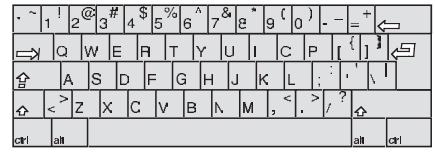
Si le système BSD sera seul sur la machine, il n'y a aucun problème : on se contente de construire un *disklabel* avec le programme d'installation. Le *disklabel* est ensuite inscrit sur le disque, et le tour est joué. Les choses se corsent si l'on souhaite faire cohabiter plusieurs systèmes d'exploitation sur le même disque : le système de partitionnement adopté doit en effet être compatible avec tous les systèmes présents.

Un bon nombre de systèmes Unix utilisent également le *disklabel*. La cohabitation est alors extrêmement facile, et ce *disklabel* servira de table de partitions pour tous. Mais la vie n'est pas toujours si simple : dans certains cas, les différents systèmes utilisent différents formats de *disklabel* tout en exigeant qu'ils soient placés au même endroit. C'est par exemple le cas de SunOS sur sparc. Dans ce cas le système BSD présente à l'utilisateur un *disklabel* à la BSD, et il le traduit en *disklabel* Sun au moment de l'écrire sur le disque.

Dans le cas de la cohabitation avec un système n'utilisant pas de *disklabel*, comme MacOS sur Macintosh ou Windows sur compatible PC, le système d'exploitation avec lequel il faut cohabiter dispose de sa propre table de partitions : deux tables sont alors nécessaires. Le système BSD construit un *disklabel* initial en se basant sur la partition du système déjà installé. Il faut donc d'abord procéder au partitionnement de l'autre système avant de compléter le *disklabel* pour BSD.

EN PRATIQUE **Un problème de clavier ?**

Il se peut que le programme d'installation ne vous propose pas de configurer le clavier. Si votre clavier est un clavier français (**AZERTY**), vous devrez alors tâtonner pour retrouver les emplacements des touches du clavier américain (**QWERTY**).
Un plan du clavier **QWERTY** vous épargnera alors une crise de nerfs. Mais ne vous inquiétez pas : à la fin de l'installation, vous pourrez configurer correctement le clavier.



Cohabitation avec MacOS

Pour aborder un cas concret, dans la cohabitation de NetBSD avec MacOS, il faut mettre en place une table de partitions Apple (*Apple Partition Map*) en plus du *disklabel*. Ceci peut se faire depuis MacOS avec les outils de formatage, ou bien

PLUS LOIN **Les noms des disques**

Les disques sont désignés par le nom du fichier spécial du répertoire /dev (*devices*, ou périphériques), qui permet d'y accéder. Chacun de ces fichiers permet de manipuler un disque comme s'il s'agissait d'un gros fichier. Les conventions de noms diffèrent beaucoup selon le système d'exploitation utilisé ; on découvrira les noms retenus dans les messages de diagnostic produits par le noyau lors du démarrage du système. Sur beaucoup de systèmes Unix, on pourra les relire grâce à la commande **dmesg | more**. Les lignes concernées sont comme celles-ci, obtenues sur un PowerMacintosh sous NetBSD :

```
sd0 at scsibus1 target 0 lun 0: <SEAGATE, ST52160N, 0285> disk fixed
sd0: 2069 MB, 6536 cyl, 4 head, 162 sec, 512 bytes/sect x 4238282 sectors
sd1 at scsibus1 target 1 lun 0: <IBM, DPES-31080, S31S> disk fixed
sd1: 1034 MB, 4903 cyl, 4 head, 108 sec, 512 bytes/sect x 2118144 sectors
cd0 at scsibus1 target 3 lun 0: <SONY, CD-ROM CDU-8005, 1.0j> cdrom removable
```

Nous disposons ici de `sd0`, disque Seagate de 2 Go, et de `sd1`, disque IBM d'1 Go. Le numéro correspond à l'ordre de découverte du disque, `sd1` deviendrait donc `sd0` en l'absence de `sd0`.
Sous NetBSD et OpenBSD, les noms de disques commencent par `sd` pour les disques SCSI et `wd` pour les disques IDE. Les versions récentes de FreeBSD utilisent plutôt respectivement `da` et `ad`.

depuis NetBSD avec le programme d'installation standard (qui utilise pour cela la commande **pdisk**). Une fois la table de partition Apple installée, le programme d'installation peut faire le partitionnement BSD avec le *disklabel*.

Cohabitation avec Windows

Autre cas concret : l'installation de n'importe quel système BSD sur un compatible PC, en cohabitation avec Windows. C'est un cas particulier, car Windows utilise la table de partitions de la zone d'amorçage maître (*Master Boot Record*, ou MBR), qui ne permet pas d'utiliser plus de quatre partitions. Cette limite est trop basse pour beaucoup d'utilisations, et l'astuce consiste à partitionner les partitions. Le programme d'installation propose d'abord un partitionnement au sens du MBR, où l'on crée une seule partition pour le système BSD. Il propose ensuite de composer une table de partitions à l'intérieur de la partition BSD du MBR en proposant un *disklabel* initial, construit à partir du MBR.

SUR LES AUTRES UNIX **GNU/Linux et le partitionnement**

GNU/Linux n'utilise pas de *disklabel* BSD. Pour sa cohabitation avec Windows, il doit faire face au même problème de limitation du nombre de partitions disponibles dans le MBR. Il le résout de manière différente, en y créant des partitions étendues. Ces partitions sont elles-mêmes découpables chacune en quatre partitions, avec des tables au même format que celle du MBR.

Cohabitation avec MacOS sur mac68k

Dans certaines situations, la cohabitation avec un autre système est obligatoire. C'est par exemple le cas des Macintosh ancienne génération, à base de processeur 680x0 : aucun des deux systèmes BSD proposés sur une telle architecture (NetBSD et OpenBSD) ne sait démarrer directement, faute de documentation sur le démarrage de ces machines. Il faut donc démarrer sous MacOS et exécuter une petite application qui charge le noyau BSD et l'invoque en faisant disparaître

CULTURE **NetBSD sur PowerMacintosh : OpenFirmware**

Sur les PowerMacintosh produits après 1995, le système assurant le démarrage de la machine est connu sous le nom d'OpenFirmware. C'est un système de démarrage assez répandu, présent aussi sur les stations Sun et IBM. Son comportement est standardisé par la norme IEEE 1275.

Sa grande force est de proposer un environnement de programmation reposant sur le langage Forth pour le développement de pilotes de démarrage indépendants de la plateforme. Les programmes en Forth sont compilés en F-Code, format binaire indépendant du processeur, interprété ensuite par une machine virtuelle – exactement comme la machine virtuelle Java interprète le *bytecode* Java.

On peut ainsi produire des pilotes pour les cartes réseau, disque, vidéo, etc., indépendants du processeur, de l'architecture de la machine et du système d'exploitation à initier. Visant à démarrer le système, ces pilotes peuvent être placés sur les cartes ou dans une mémoire non volatile de l'unité centrale.

Le lecteur curieux pourra en apprendre plus sur OpenFirmware à l'adresse : http://playground.sun.com/1275. Cet environnement est très expressif, mais il restera inexploitable pour ceux qui n'auront pas su s'imprégner de la syntaxe torturée du Forth. En voici un bref exemple :

```
0> cr 10 0 do I . loop
0 1 2 3 4 5 6 7 8 9 A B C D E F  ok
0>
```

Malheureusement, les premières implémentations Apple d'OpenFirmware souffraient de nombreux bogues (dont l'absence de la commande `help`). Ces problèmes furent corrigés à partir des machines dites « *new world* », qui emploient un OpenFirmware de version 3.0 ou ultérieure. Pour les machines plus anciennes, il faut mettre en œuvre de nombreux contournements de bogues dépendants de la version d'OpenFirmware pour pouvoir démarrer NetBSD. La procédure d'installation relève du casse-tête, et il est vivement recommandé de lire très attentivement les notes d'installation si l'on souhaite éviter les pires complications. Certaines mauvaises manipulations peuvent même mettre la machine hors service.

OpenBSD, pour sa part, ne fonctionne que sur les machines « *new world* ». La procédure est donc plus simple, puisqu'elle dépend moins du modèle.

MacOS de la mémoire de la machine. Si un seul disque est disponible, il faut donc y avoir, en plus de la partition BSD, une partition HFS pour démarrer MacOS.

Cohabitation avec Windows CE sur assistant personnel (PDA)

Dans le cas des installations de NetBSD sur des assistants personnels (PDA), à l'origine prévus pour Windows CE, la cohabitation est encore obligatoire : Windows CE étant enregistré en mémoire permanente (ROM), il faut démarrer la machine sous ce système. Là encore, une petite application permet de démarrer NetBSD. Cette application, pour être accessible par Windows CE, sera placée sur une partition DOS, qui devra cohabiter avec le partitionnement NetBSD. Windows CE fonctionnant avec le MBR comme les différents Windows sur compatibles PC, le partitionnement se fait exactement comme sur un PC. Seule différence : il faut disposer d'une partition DOS pour le démarrage, alors qu'elle n'est pas nécessaire sur PC.

Partitionnement

Il est enfin temps d'évoquer le partitionnement du système BSD. Il faut mettre en place au minimum une partition pour la racine du système de fichiers, et une partition de *swap* est vivement conseillée. Dans certains cas particuliers, on peut s'en passer, mais la machine risque alors de planter si elle tombe à court de mémoire vive.

On peut mettre en place d'autres partitions, pour /var ou /usr. Séparer /usr permet de le monter en lecture seule quand on n'a plus besoin de le modifier. On évite ainsi toute fausse manipulation destructrice, et gagne du temps en cas de redémarrage impromptu (dû par exemple à une coupure brusque de courant). En effet, les volumes montés en lecture seule n'auront pas besoin d'être contrôlés au redémarrage : aucune opération d'écriture n'ayant pu être interrompue, le système de fichiers ne peut pas avoir été corrompu.

On peut aussi prévoir une partition séparée pour /var, lieu de résidence de tous les fichiers souvent modifiés. On y trouvera les fichiers de journalisation (journaux, ou *logs*), des fichiers temporaires, les files d'attente du courrier électronique, et bien d'autres choses encore. Séparer /var de la racine permet d'assurer que tout débordement de file d'attente ou tout gonflement des journaux ne saturera pas la racine. C'est une pratique saine sur les serveurs.

Dans leur installation comme dans leur utilisation quotidienne, les systèmes BSD désignent les partitions par des lettres. `a` représente la première partition, `b` la deuxième. `c` et dans certains cas `d` sont des cas particuliers : pour NetBSD, sur les plateformes disposant d'un MBR, `d` est une pseudo-partition représentant en fait l'intégralité du disque. `c` représente alors l'intégralité de la partition NetBSD dans le MBR. En l'absence de partitionnement au sens du MBR, si tout le disque est attribué au système BSD, `c` et `d` sont identiques. Les lettres suivantes représentent

ATTENTION **Taille des partitions**

Les partitions doivent respecter des contraintes de taille : trop petites, elles ne pourront pas accueillir tous les composants du système d'exploitation. Trop grandes, elles risquent d'occuper inutilement de l'espace alors que vous en manquerez sur une autre partition.

Les tailles minimales des partitions varient beaucoup d'un système et d'une version à l'autre à l'autre. Consultez les notes d'installation pour obtenir des informations précises.

Dans le doute, si vous montez une machine qui sera un terrain d'expérimentation et qui ne doit pas devenir un serveur en production, vous pouvez envisager de tout placer sur la racine et de ne pas séparer /usr, /var ou /home. Vous serez ainsi certain de ne pas gâcher de place et pourrez expérimenter les cas où un partitionnement plus fin aurait été souhaitable.

CULTURE **Les fichiers de log**

Les fichiers de *log*, généralement situés sous /var/log, sont les journaux des événements que l'on souhaite consigner. Par exemple, le fichier authlog conserve une trace de toutes les connexions à la machine. maillog concerne les messages de courrier électroniques transitant sur le système. messages, fichier fourre-tout, regroupe des messages d'erreur ou avertissements en provenance de nombreux sous-systèmes.

C'est le processus **`syslogd`** qui est chargé de consigner les événements qui lui sont signalés. Le fichier /etc/syslog.conf, configuré par l'administrateur, indique où stocker chaque type d'événement.

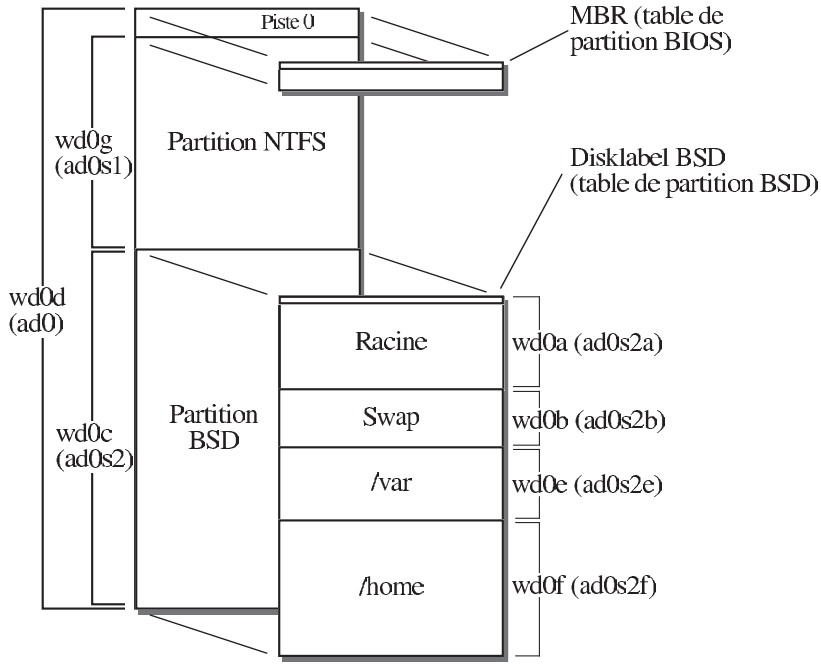


Figure 4–1 Cohabitation du partitionnement par le MBR et du partitionnement BSD (convention de noms NetBSD, suivie entre parenthèses du nom FreeBSD)

ATTENTION **Partitions Windows**
Cela va sans dire, mais mieux en le disant : les lettres désignant les partitions sous BSD n'ont a priori rien à voir avec les lettres des lecteurs sous Windows.
Il est possible qu'une partition vue comme `wd0e` sous BSD soit appelée `E:` sous Windows, mais cela ne serait alors qu'une coïncidence.

les autres partitions : `e` pour la troisième, `f` pour la quatrième et ainsi de suite. La figure 4.1 donne un bon aperçu du rôle des partitions `c` et `d`.

Sur les plateformes NetBSD n'utilisant pas le MBR, et sous OpenBSD, `c` représente tout le disque, `d` la troisième partition, `e` la quatrième, et ainsi de suite.

Pour FreeBSD, la partition `d` n'a jamais de rôle particulier, et `c` représente soit la totalité de la partition MBR allouée à FreeBSD, soit la totalité du disque en l'absence de MBR.

Si plusieurs systèmes BSD doivent partager le même disque (double amorçage, ou disque externe voyageant d'une machine à l'autre), souvenez-vous qu'ils partagent tous le même *disklabel*. Il est donc préférable de ne pas utiliser de partition `d` sur tout disque utilisé alternativement sur un système donnant un sens spécial à `d` et sur un système où cette partition est quelconque.

En général, la partition a est utilisée pour la racine. Sur beaucoup de plateformes, il sera d'ailleurs impossible de démarrer si la racine ne s'y trouve pas ou si cette partition n'est pas placée au début de la partition BSD. De même, b est souvent réservé à la pagination (*swap*).

IMPORTANT **De la bonne nomenclature des partitions**

Le nom des fichiers spéciaux de /dev donnant accès aux partitions est un domaine où les Unix sont très versatiles. On compte presque autant de conventions que de systèmes. Sur un Unix inconnu, la commande **df** vous indiquera les partitions déjà montées, ce qui pourra vous aider à comprendre les conventions de noms adoptées par le système.

Récapitulons les cas qui nous intéressent, et mentionnons-en quelques autres à titre de comparaison.

La figure 4.2 récapitule l'attribution de noms sous NetBSD et OpenBSD. Les partitions `c`, et parfois `d`, ont un usage spécial :

- Pour NetBSD, en présence d'un partitionnement MBR sur les plateformes qui l'utilisent, `c` est l'ensemble de la partition NetBSD du disque, et `d` représente l'ensemble du disque.
- Pour NetBSD, sur une plateforme utilisant habituellement le MBR sans avoir fait de partitionnement MBR, `c` et `d` sont identiques et représentent la totalité du disque.
- Dans tous les cas pour OpenBSD, et pour NetBSD sur les architectures sans MBR, `c` représente la totalité du disque et `d` est une partition quelconque, disponible pour l'utilisateur.

Il arrive que certains programmes acceptent des noms de disque sans partition, comme par exemple `sd0`. Selon le contexte, ils utilisent en réalité /dev/sd0c ou /dev/sd0d.

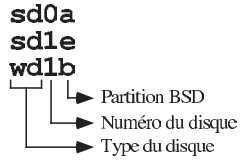


Figure 4–2 Nom des partitions sous NetBSD et OpenBSD

FreeBSD a utilisé un système semblable à celui de NetBSD et OpenBSD, désormais remplacé par la notation décrite dans la figure 4.3 : le type du disque (`ad` ou `da`), son numéro, un `s` comme « *slice* » (partition), le numéro de partition MBR (la numérotation commence à `1` pour les partitions du MBR, et à 5 pour les partitions faites dans une partition étendue), et la lettre de la partition BSD.

La partition `d` est une partition comme les autres, et `c` représente la totalité de la partition MBR. Contrairement à NetBSD et OpenBSD, il existe un fichier /dev/ad0 qui donne accès à tout le disque `ad0`.

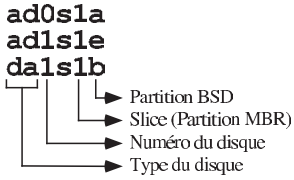


Figure 4–3 Nom des partitions sous FreeBSD

Il est intéressant de comparer ces conventions avec celles d'autres systèmes. La figure 4.4 décrit ce qui se passe avec Linux. Les disques SCSI s'y appellent `sd` et les disques IDE, `hd`. Les disques sont repérés par des lettres (en commençant par `a`), et les partitions, par des chiffres (à partir de `1`). De même que sous FreeBSD, on trouve des fichiers spéciaux représentant tout un disque, comme par exemple /dev/hda.

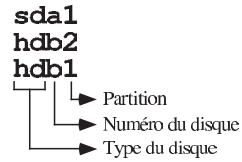


Figure 4–4 Nom des partitions sous Linux

Solaris utilise un système plus complexe, pensé dès le départ pour l'exploitation de machines contenant de nombreux disques. Il est décrit dans la figure 4.5. Pour aider l'administrateur, le nom des partitions contient dans l'ordre les numéros du contrôleur de disques, de la cible sur le bus (pour un bus SCSI c'est l'ID SCSI), du numéro d'unité logique (LUN), et de la partition. Les lettres `c`, `t`, `d` et `s` précèdent respectivement chacun de ces numéros.

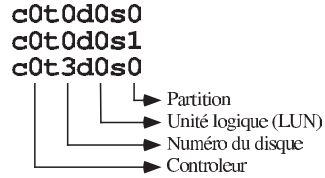


Figure 4–5 Nom des partitions sous Solaris

Enfin, la figure 4.6 indique la manière dont MacOS X s'y prend. Les disques et partitions sont repérés par des numéros. On commence à `0` pour les disques, et à `1` pour les partitions, la partition `1` étant réservée à la table des partitions Apple. Comme sous FreeBSD et Linux, on trouve des fichiers spéciaux pour manipuler le disque entier, tels que /dev/disk0.

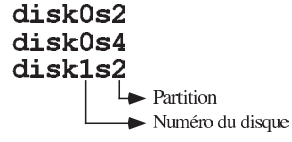


Figure 4–6 Nom des partitions sous MacOS X

ALTERNATIVE **Console série**

La console série permet de se passer d'écran et de clavier sur une machine. On connecte à son port série un terminal de type vt100 ou une autre machine, sur laquelle on emploiera un émulateur de terminal.
Certaines machines sont dépourvues d'écran et de clavier (c'est par exemple le cas des cartes embarquées). Il faut donc y recourir à la console série. Cette dernière est aussi appréciée pour les serveurs disposés dans des baies (*racks*) : on économise ainsi de l'espace, et l'administrateur qui a besoin de la console vient se brancher sur le port série du serveur avec un portable. L'auteur utilise le Jornada du chapitre 1 à cet usage.
Malheureusement, la console série ne permet pas de contrôler totalement la machine si la ROM ne sait utiliser qu'un écran et un clavier. C'est par exemple le cas sur les PC, où l'absence de clavier peut même empêcher le démarrage. En revanche, la plupart des stations de travail utilisent spontanément le port série comme console lorsqu'aucun écran n'est branché.

B.A.-BA **La ROM**

La ROM (*Read Only Memory*, ou mémoire permanente) de la machine est une partie de la mémoire en principe inaltérable. Elle contient un programme exécuté au démarrage, dont le rôle est d'initialiser un minimum la machine puis de charger l'amorce du système d'exploitation.
Sur un compatible PC, la ROM est communément connue sous le nom de BIOS (*Basic Input/Output System*, ou système d'entrées/sorties basiques). Sur la plupart des autres architectures, on parle du *firmware*. On retrouve certains *firmwares* sur plusieurs types de machines différentes, comme ARC sur stations Silicon Graphics et Alpha de Digital. Parfois, le *firmware* est le même, mais porte un nom diffèrent : OpenFirmware sur PowerMacintosh et station IBM, et BootPROM sur station Sun.

Amorçage

L'installation de l'amorce par le programme d'installation est une étape assez discrète, qui ne donne pas forcément lieu à une configuration. Dans le cas le plus simple, le programme d'installation ne demande rien et installe une amorce pour que le système puisse démarrer sur la partition racine.

Le programme d'installation de NetBSD propose tout de même le choix entre une amorce standard et une amorce série (*serial boot blocks*). Il s'agit de choisir si la console doit être envoyée sur un écran ou sur un port série. Le cas des configurations en double amorçage donne également lieu à quelques choix lors de l'installation de l'amorce. Sur FreeBSD, il est possible de choisir, à l'installation, d'installer ou non une amorce, qu'on peut ensuite configurer pour qu'elle envoie la console sur le port série.

Double amorçage

Le programme d'installation met en place une amorce sur la partition racine du système BSD. Sur une configuration en double amorçage (*dual-boot*), l'autre système aura aussi une amorce sur sa partition. Reste à sélectionner sur quel système se fera le démarrage.

Il y a plusieurs approches. Certaines architectures sauront sélectionner la partition de démarrage depuis la ROM. C'est le cas des PowerMacintosh, sur lesquels on peut même configurer OpenFirmware pour afficher un menu de démarrage proposant le choix entre différents systèmes.

Sur d'autres architectures, la ROM ne sait pas faire grand-chose, et c'est aux amorces de gérer le double amorçage. C'est le cas sur PC : chaque système a sa partition au sens du MBR, avec son amorce au début. Le MBR indique dans sa table de partitions une partition dite « active », qui sera utilisée pour démarrer.

Le MBR contient aussi une amorce, premier programme exécuté par la ROM. En temps normal, l'amorce du MBR se contente d'invoquer l'amorce de la partition active, mais on peut installer dans le MBR une amorce proposant un menu des systèmes à démarrer. Ce menu peut encore être proposé par l'amorce de la partition active ; c'est ainsi que procèdent Windows NT et ses descendants.

Le programme d'installation peut proposer l'élaboration d'un menu de démarrage, ou bien vous pouvez compter sur l'autre système pour présenter un menu – c'est à vous de choisir.

Où sont les archives ?

Il faut maintenant installer les fichiers qui composent le système d'exploitation à proprement parler. Ces derniers sont fournis sous la forme d'archives au format **`tar`**, compressées avec **`gzip`** (fichiers .tgz) pour NetBSD et OpenBSD, et sous forme de fragments d'archives **`tar`** compressés avec **`gzip`** pour FreeBSD. Ces fragments sont suffisamment petits pour tenir sur des disquettes le cas échéant.

ALTERNATIVE **Quelques autres amorces**

Linux utilise le programme LILO (*LInux LOader*, ou chargeur de Linux) pour l'amorce. Sa configuration se fait dans /etc/lilo.conf, et on la valide par la commande **`lilo`**.
Windows NT et ses successeurs ont une amorce appelée NTLDR, qui utilise le fichier boot.ini, situé à la racine, pour le menu de démarrage. On peut facilement le configurer pour ajouter un autre système : il faut placer à la racine de la partition Windows un fichier contenant une copie de l'amorce du système à démarrer (les 512 premiers octets de la partition). Sous NetBSD ou OpenBSD, on obtient ce résultat par la commande **`dd if=/dev/rwd0a of=amorce bs=512 count=1`**). Il suffit ensuite d'ajouter la ligne suivante au boot.ini :

```
C:\amorce="BSD Unix"
```

Enfin, il est intéressant de mentionner GRUB (*GRand Unified Bootloader*, ou grand chargeur d'amorçage unifié). C'est une amorce générique, prévue pour fonctionner avec le plus grand nombre de systèmes possible. GRUB gère les problèmes de *netboot* ou de menu de démarrage.

Lors de l'installation, la machine doit pouvoir accéder à l'ensemble de la distribution, par exemple via FTP ou NFS, ou encore sur un CD-ROM. Si l'on opte pour un accès à la distribution par FTP ou NFS, il faut évidemment une connexion au réseau. Cette voie est la plus simple : il suffit alors de démarrer le noyau d'installation, souvent grâce à des disquettes de démarrage, et de disposer d'une connexion à un réseau.

Sur tous les BSD, le noyau d'installation reconnaît beaucoup de cartes réseau Ethernet. Mais au moment où sont écrites ces lignes, l'installation via un modem téléphonique (RTC) n'est possible que sur FreeBSD, et aucun BSD n'est capable de faire une installation via une connexion ADSL (sauf bien sûr dans le cas où la machine est connectée sur un réseau local connecté à l'ADSL par une autre machine).

EN PRATIQUE **Installation modem ou ADSL**

Si le système retenu n'est pas capable de faire une installation par modem ou par ADSL, cela n'implique en rien qu'il est incapable de les exploiter une fois installé, rassurez-vous.

Finitions

La dernière étape consiste à mettre en place un minimum de configuration pour que la machine puisse fonctionner : sélection du fuseau horaire pour l'heure locale, et choix du mot de passe du compte de root.

PIÈGE À C... **Mot de passe et clavier QWERTY**

Si vous avez installé le système avec un clavier AZERTY fonctionnant en QWERTY, prenez garde à votre choix de mot de passe. Vous risquez en effet d'être incapable de le taper correctement quand le clavier sera bien configuré.

Installation sans le programme d'installation

Sur certaines plateformes exotiques, la procédure d'installation n'est pas aussi avancée que cela. En général, ces plateformes sont dépourvues de notes d'installation, car un malheur n'arrive jamais seul. L'installation dans ce cas n'est pas vraiment à portée du débutant isolé. Deux pistes sont envisageables.

La première est de démarrer la machine à installer à travers le réseau, si elle en est capable. On prépare sur un serveur un système de fichiers contenant les

outils d'installation, et on démarre la machine à installer avec un noyau configuré pour monter la racine du système de fichiers depuis ce serveur. Les services à y configurer sont DHCP ou BootP, éventuellement TFTP, et NFS. Leur configuration est abordée au chapitre 8.

La deuxième voie est de démonter le disque de la machine sur laquelle on souhaite procéder à l'installation et de le placer dans une machine sur laquelle on dispose des outils d'installation. Cette machine hôte devra probablement être un système BSD car certains outils de formatage peuvent s'avérer délicats à trouver sur des plateformes très différentes.

Une fois en selle, on commence par partitionner le disque sur lequel on souhaite installer. En cas de cohabitation avec un autre système, il faudra peut-être modifier sa table de partitions, avec **`fdisk`** pour le MBR des machines qui emploient Windows, ou **`pdisk`** pour celles qui fonctionnent d'habitude sous MacOS.

Si l'on doit utiliser **`fdisk`**, on peut buter sur certaines questions sibyllines, comme la valeur numérique du « SysId » à attribuer à la partition. Dans le MBR, ce nombre sert à caractériser le type de chaque partition. Exemples :

Système	SysId (déc.)	SysId (hex.)
NetBSD	169	0xa9
FreeBSD	165	0xa5
OpenBSD	166	0xa6
Linux ext2fs	131	0xa6
FAT16	4 ou 6	0x4 ou 0x6
FAT32	11 ou 12	0xb ou 0xc
NTFS	7	0x7
Partition étendue	5	0x5

Ensuite, il faut mettre en place un *disklabel* avec la commande éponyme (**`disklabel`**). Elle peut être invoquée de différentes manières, selon qu'on veut lire ou écrire sur le *disklabel*. La syntaxe différant légèrement d'un système BSD à l'autre, il est judicieux de lire la page **`man`** de cette commande. Pour invoquer votre éditeur favori et y créer un nouveau *disklabel* qui sera ensuite inscrit sur le disque, tapez **`disklabel -I -r -e sd0`** sous NetBSD ou **`disklabel -e -r sd0 auto`** sous FreeBSD (en remplaçant sd0 par le nom du disque).

Point suivant, le formatage des partitions BSD (racine, /var, /usr...), réalisé par la commande **`newfs`**. Si l'on installe depuis une autre machine par transplantation de disque dur, il faut éventuellement utiliser l'option **`-B`**, qui permet de préciser si le formatage doit être petit boutiste (*little endian*) ou gros boutiste (*big endian*) ; voir à ce sujet la page **`man`** de **`newfs`**.

SUR LES AUTRES UNIX **Pour formater**

Sur certains Unix (et en particulier sur les anciennes versions des BSD), c'est la commande **`mkfs`** qui permet de formater. Sous GNU/Linux, il existe encore une commande **`mkswap`** qu'il faut utiliser pour préparer les partitions de pagination (*swap*) avant usage – mais la plupart des systèmes Unix n'ont pas besoin d'une telle préparation.

CULTURE **DHCP, BootP, TFTP et NFS**

Ces acronymes désignent des protocoles utilisés entre autres pour démarrer une machine via le réseau. *Dynamic Host Configuration Protocol* (DHCP), et *Boot Protocol* (BootP), son ancêtre, permettent à une machine qui démarre de découvrir sa configuration réseau et à quelle machine s'adresser pour continuer le démarrage par TFTP ou NFS.

Trivial File Transfer Protocol (TFTP) est un protocole de transfert de fichiers simplifié le plus possible, qui permet à une machine de télécharger son noyau ou divers fichiers. Des engins tels que les routeurs Cisco ou les terminaux X en font un usage abondant pour télécharger leurs systèmes d'exploitation ou leurs fichiers de configuration.

Enfin, *Network File System* (NFS) est un protocole permettant de monter un système de fichiers d'une machine distante à travers le réseau. NFS est le protocole natif de partage de fichiers sous Unix, et il sert en beaucoup d'autres occasions, comme par exemple sur parc de machines banalisées, pour que l'utilisateur retrouve son répertoire personnel sur toutes les machines, de manière transparente.

EN PRATIQUE **`fdisk` et le SysId**

Sous NetBSD, la commande **`fdisk -l`** permet d'obtenir la liste des SysId connus et les types de partition auxquels ils correspondent.

PLUS LOIN **L'éditeur favori**

L'éditeur favori est indiqué dans la variable d'environnement `EDITOR`. En son absence, la plupart des programmes utilisent /usr/bin/vi. Pour spécifier /usr/pkg/bin/emacs comme éditeur favori avec **`sh`**, **`ksh`**, ou **`bash`**, tapez

```
$ EDITOR=/usr/pkg/bin/emacs
$ export EDITOR
```

Sous **`csh`** ou **`tcsh`**, il faudra taper :

```
$ setenv EDITOR /usr/pkg/bin/emacs
```

Après avoir formaté les systèmes de fichiers, il faut les monter dans l'arborescence à l'aide de la commande **mount**. /mnt est en général utilisé pour ce genre de montages provisoires. Il faut ensuite décompacter les archives de la distribution avec la commande **tar**, dont on utilisera l'option **-p** pour s'assurer que les permissions des fichiers ne seront pas perdues (**tar** ne l'utilise pas par défaut sur tous les systèmes).

Dernière touche, la création des fichiers spéciaux de /dev, à l'aide du script **/dev/MAKEDEV** (placé là par **tar**), la création d'un fichier /etc/fstab avec des renseignements pertinents, et l'installation de l'amorce du système. Cette dernière opération dépend énormément du système. Sous NetBSD et OpenBSD, **installboot** est la commande à utiliser ; sous FreeBSD, c'est **disklabel**. Exemple sur NetBSD/i386 pour installer une amorce normale, utilisant la console pour afficher et le disque pour démarrer :

```
# installboot /usr/mdec/biosboot.sym /dev/rsd0a
```

Sous NetBSD/i386, **installboot** créera l'amorce secondaire /boot à partir de biosboot.sym. Sous OpenBSD/i386, il faut copier l'amorce secondaire sur le disque avant d'utiliser **installboot** :

```
# mount /dev/sd0a /mnt
# cp /usr/mdec/boot /mnt/boot
# /usr/mdec/installboot -v /mnt/boot /usr/mdec/biosboot sd0
```

Sous FreeBSD/i386, c'est la commande **disklabel** invoquée avec l'option **-B** qui installe l'amorce primaire, comme ceci :

```
# disklabel -B da0
```

L'amorce primaire est lue à partir de /boot/boot1. L'amorce secondaire /boot/boot2 doit être présente, et /boot/loader servira d'amorce tertiaire. Quant à /boot/boot0, c'est une amorce pour le MBR. Elle est normalement installée par **fdisk**, mais on peut configurer son comportement avec la commande **boot0cfg**.

L'installation manuelle n'a maintenant plus aucun secret pour vous. À part dans certains cas assez particuliers, il est assez rare d'avoir à y recourir pour une installation. En revanche, ces opérations pourront servir dans le cas d'une intervention en réparation.

CULTURE **Petits et gros boutistes**

Il s'agit de l'ordre dans lequel la machine stocke les mots de deux octets dans la mémoire. Les gros boutistes commencent par l'octet de poids fort : `0x1234` est placé en mémoire à l'endroit, comme suit : `0x1234`. Les petits boutistes commencent par l'octet de poids faible et stockent la valeur à l'envers : `0x3412`. La plupart des processeurs sont gros boutistes ; les exceptions sont l'ARM, l'Alpha, le VAX, le 80x86 d'Intel, et ses clones. Les processeurs récents sont capable de comprendre les deux conventions. Le choix se fait au démarrage et on n'en change plus une fois que le système d'exploitation fonctionne.
En anglais, on parle respectivement de machines *little endian* ou *big endian*. La traduction en « petit boutistes » ou « gros boutistes » fait référence aux mangeurs d'œufs des voyages de Gulliver : certains les cassent par le petit côté, d'autres par le gros côté.

ATTENTION **Pas de MAKEDEV ?**

À partir de FreeBSD 5.0, un dispositif appelé devfs prends en charge la création dynamique d'un /dev contenant les fichiers adéquats, il n'est donc plus nécessaire d'utiliser un script **MAKEDEV** pour créer les fichiers du /dev.
À l'heure où sont écrites ces lignes, NetBSD n'a pas encore adopté de devfs. Par contre, il saura créer de lui-même les fichiers du /dev au moment du démarrage, s'ils ne sont pas présents. L'usage du **MAKEDEV** n'est donc pas obligatoire, mais on gagne en performances en l'utilisant.

PLUS LOIN **fstab**

Le fichier /etc/fstab indique quelles partitions monter au démarrage, et où les monter. En voici un exemple :

```
/dev/sd0a       /       ffs     rw 1 1
/dev/sd0b       none    swap    sw 0 0
/dev/sd0g       /var    ffs     rw 1 1
/dev/sd1a       /mnt    ffs     rw 1 1
kern            /kern   kernfs  rw 0 0
proc            /proc   procfs  rw 0 0
```

Chaque ligne contient dans l'ordre : le fichier spécial donnant accès au disque, le point de montage (c'est-à-dire le répertoire de l'arborescence du système de fichiers où cette partition sera placée), le type de système de fichiers, les options de montage (`rw` signifie *Read/-Write* : montage en lecture/écriture). Les deux dernières colonnes sont des options pour les commandes **dump** et **fsck**.
Le nom du fichier fstab peut varier légèrement selon les systèmes : sur Solaris, il s'appelle par exemple vfstab.

Que faire maintenant ?

Une fois l'installation terminée, la machine redémarre, et si tout s'est bien passé, elle propose une invite de connexion (*login*). Exemple sous NetBSD :

```
Setting tty flags.
Starting network.
Hostname: violette
Configuring network interfaces: mc0.
add net default: gateway 10.0.12.1
Adding interface aliases:
Building databases...
Starting syslogd.
Checking for core dump...
savecore: no core dump
Clearing /tmp.
Checking quotas: done.
Setting securelevel: kern.securelevel: 0 -> 1
Starting virecover.
starting local daemons:.
Updating motd.
Starting inetd.
Starting cron.
Fri May  9 07:26:50 UTC 2003

NetBSD/macppc (violette) (tty00)

login:
```

À l'invite de *login*, tapez **`root`**, puis tapez le mot de passe que vous aviez donné au programme d'installation. La machine invoque un shell, et vous voilà l'heureux administrateur d'une machine Unix à configurer.

Où est passée l'interface graphique ?

Si vous venez du monde Windows ou même des distributions GNU/Linux les plus à la mode, le fait que la machine ne démarre qu'en mode texte peut perturber quelque peu. Tout est normal : l'installation n'a pas échoué ; les BSD n'invoquent pas d'interface graphique tant que celle-ci n'est pas configurée. Voyez cela comme un avantage : vous allez apprendre comment celle-ci fonctionne, et si un jour elle refuse de fonctionner, vous saurez comment procéder pour réparer la machine. Si vous êtes trop impatient pour attendre le chapitre 7, et si vous pensez être très chanceux, essayez la manière rapide : tapez la commande **`XFree86 -configure`**, installez le fichier de configuration qu'elle produira, avec **`cp XF86Config.new /etc/X11/`**, et invoquez enfin **`startx`**.

Si cela ne fonctionne pas, n'en déduisez pas que le mode graphique ne fonctionnera pas sur la machine ; cela signifie simplement que son installation sera un peu plus compliquée que cela.

Les prochaines étapes

Vous pouvez ajouter des utilisateurs (chapitre 6), configurer le réseau ou l'interface graphique (chapitre 7), monter un pare-feu (chapitre 10), installer des logiciels, comme par exemple votre shell favori ou les bureaux GNOME et KDE (chapitre 11), configurer des services tels qu'un serveur Web ou une messagerie (chapitre 12)... Le travail ne manque pas, il faut juste décider par où commencer.

EN CAS DE COUP DUR

Si vous n'arrivez pas jusque là, c'est que quelque chose s'est mal passé. L'installation a peut-être tourné court ? Le chapitre suivant aborde les problèmes de démarrage ; vous pouvez vous y reporter pour trouver une solution à vos problèmes. En désespoir de cause, les listes de diffusion des projets BSD offriront une assistance technique aux malheureux dont le système refuse de démarrer.

ASTUCE **Un environnement un peu plus confortable**

Si vous souhaitez utiliser un clavier français, et si la procédure d'installation ne vous a pas donné l'occasion de le configurer correctement, le clavier est encore configuré à l'américaine, en **QWERTY**. C'est pénible, mais pas insurmontable.

Sous NetBSD et OpenBSD, on passe en clavier français en tapant dans le shell la commande **wsconsctl -w encoding=fr**. Sur NetBSD, pour obtenir automatiquement cette configuration au prochain démarrage, ajoutez une ligne

```
wscons=YES
```

dans /etc/rc.conf, et une ligne

```
encoding fr
```

dans /etc/wscons.conf.

Sur OpenBSD, il suffit d'ajouter la ligne suivante dans le fichier /etc/wsconsctl.conf.

```
encoding=fr
```

Pour FreeBSD, tapez **kbdcontrol -l fr.iso.acc**. La configuration automatique au démarrage s'obtiendra en ajoutant la ligne suivante dans /etc/rc.conf :

```
keymap="fr.iso.acc"
```

Outre les problèmes de claviers, l'environnement fourni par défaut sous NetBSD et OpenBSD n'est pas forcément très convivial. Si vous êtes un habitué d'Unix, vous vous attendez sans doute à pouvoir éditer la ligne de commande en cours de saisie, rappeler les commandes précédentes avec les touches fléchées, et bénéficier de la complétion (quand dans le shell le touche de tabulation complète automatiquement les noms de commandes, de fichiers ou de répertoires). Essayez ceci (attention **^I** est un accent circonflexe suivi de la lettre **I**, et non pas la combinaison de touches **Control**+**I**) :

```
# ksh
# set -o emacs
# bind '^I'=complete-file;
```

Si le terminal est un peu inhabituel, il se peut que les touches fléchées et la tabulation ne fonctionnent pas. Il faudra alors utiliser :

- **Control**+**P** pour rappeler la commande précédente (*previous*)
- **Control**+**N** pour rappeler la commande suivante (*next*)
- **Control**+**B** pour se déplacer à gauche (*backward*)
- **Control**+**F** pour se déplacer à droite (*forward*)
- deux fois **Échap** pour la complétion

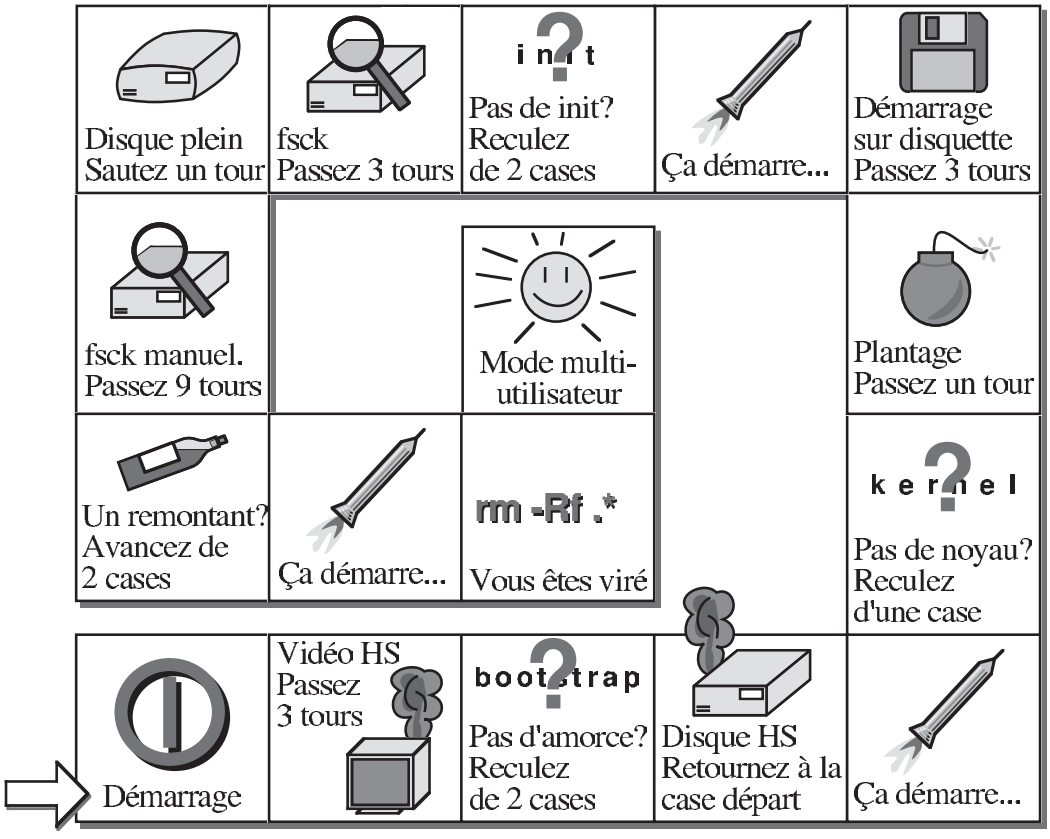
Pour configurer automatiquement tout cela au démarrage, changez votre shell en **ksh** (vous apprendrez comment faire dans le chapitre 6), et placez les deux lignes suivantes dans un fichier .profile situé dans votre répertoire personnel :

```
set -o emacs
bind '^I'=complete-file;
```

FreeBSD dispose dans la distribution de base de **tcsh**, qui offre ces fonctionnalités sans configuration particulière.

5

JEU DE L'OIE DE L'ADMINISTRATEUR SYSTÈME



Démarrage des systèmes Unix

Pour se sortir des situations critiques où la machine ne démarre pas correctement, il est important de maîtriser la séquence d'initialisation du système. C'est le chemin indispensable pour transformer un tas de ferraille inerte en l'indispensable serveur que tout le monde utilise et dont le fonctionnement correct justifie votre salaire d'ingénieur système.

SOMMAIRE

MOTS-CLEFS

- Amorçage
- Noyau du système
- Mode mono-utilisateur

Ce chapitre traite du démarrage de la machine. Les considérations développées sont assez génériques, et s'appliquent à la plupart des systèmes Unix. Les aspects spécifiques aux systèmes BSD seront bien entendu plus détaillés.

Petit rappel historique

Nous l'avons vu, Unix est un système à la généalogie complexe. Ceux que l'arbre du chapitre 1 n'a pas convaincus sont invités à explorer l'arbre généalogique d'Unix reconstitué par Éric Lévénez à l'adresse http://www.levenez.com/unix/. Pour le citer, la généalogie d'Unix est « une sorte de jungle amazonienne bien touffue. On pourrait aussi dessiner un plat de nouilles en fait... »

Heureusement, tous ces systèmes ont des liens de parenté, donc des comportements communs dans la plupart des situations. La famille compte trois branches principales, d'où trois comportements possibles : les UNIX System V, les Unix BSD, et GNU/Linux. Ce dernier est une ré-implémentation d'un UNIX System V en partant de zéro (*from scratch*). Il ne descend pas de System V, mais se comporte comme lui sur bien des aspects. Il sera implicitement concerné par toute référence aux UNIX System V dans la suite du chapitre, sauf mention contraire.

CULTURE **Linux hybrides**

La classification des systèmes est difficile, certaines distributions Linux reprenant des comportements BSD. C'est par exemple le cas de la Slackware.

PLUS LOIN **Accéder à la console du firmware**

Parfois, on souhaite accéder à la console du *firmware*, pour par exemple démarrer le noyau avec une option particulière, ou pour choisir le disque sur lequel démarrer. Cette opération dépend évidemment du matériel concerné. Par exemple, les PowerMacintosh invoqueront une console OpenFirmware (éventuellement sur le port série) après maintien au démarrage de la combinaison de touches **Pomme**+**Option**+**O**+**F**. Les stations Sun donnent un accès à la console si l'on enfonce la combinaison **Stop**+**A** (ou **Stop**+**Q** sur un clavier **AZERTY**).
Sur PC, pas de *firmware* avec console, mais il est possible d'interrompre l'amorce pour indiquer des paramètres de démarrage particuliers. Sur les BSD, l'amorce est par défaut configurée avec un compte à rebours de quelques secondes avant de démarrer. Sur NetBSD et OpenBSD, taper **Control**+**C** pendant le compte à rebours permet de prendre la main. On obtient le même résultat sur FreeBSD en appuyant sur **Espace**.

PLUS LOIN **Amorçage via le réseau**

Dans ce cas, la ROM sait utiliser DHCP ou BootP pour obtenir une adresse IP et trouver le serveur où télécharger l'amorce via TFTP ou NFS. On peut mettre au point des démarrages hybrides lorsque la ROM est incapable de chercher l'amorce sur le réseau, et par exemple démarrer sur disquette une amorce capable d'utiliser DHCP pour obtenir son adresse IP, puis de télécharger un noyau via TFTP.

Amorçage

Cette étape dépend beaucoup de la plateforme matérielle. La ROM de la machine charge puis exécute l'amorce du système d'exploitation qui peut être lue sur une disquette, sur un disque dur, un CD-ROM... tout est possible, à condition que la ROM sache comment y accéder.

La ROM ne sachant en général rien du formatage du disque, l'amorce est toujours stockée au même endroit, généralement au début du disque. Sur bien des machines, la ROM se contente de lire les 512 premiers octets du disque ou de la disquette, et d'exécuter ce contenu. S'il s'agit d'une amorce en bonne et due forme, le démarrage continue, sinon la machine plante.

L'amorce est souvent capable de comprendre le partitionnement du disque ainsi que les systèmes de fichiers des partitions. Elle sait donc trouver le noyau, puis l'exécuter. Bien souvent, l'amorce ou la ROM offre à l'opérateur une possibilité d'interrompre le démarrage automatique pour indiquer des options de démarrage particulières ou le nom et l'emplacement du noyau à charger. Sur les trois systèmes BSD, la page **man** `boot(8)` décrit les options disponibles dans l'amorce. On en a souvent besoin le jour où le système refuse de démarrer ; il est donc préférable de l'avoir lue avant de connaître une situation critique.

Dans certains cas, la ROM de la machine a une limite assez basse sur la taille de l'amorce qu'elle peut charger. Il est alors impossible d'y mettre tout le code nécessaire pour comprendre le système de fichiers et charger le noyau. Dans de telles situations, on utilise deux amorces. Une petite amorce primaire, que la ROM peut charger, contenant la position explicite sur le disque de l'amorce secondaire et se contentant de l'exécuter – elle n'a pas besoin pour cela de comprendre

le système de fichiers. L'amorce secondaire, plus volumineuse, est capable de trouver et charger le noyau.

Sur certaines plateformes, la ROM est capable de comprendre le partitionnement d'un disque et son système de fichiers ; on peut alors démarrer directement le noyau sans amorce. C'est très confortable, mais cela souffre d'un inconvénient majeur : il est impossible de s'écarter du partitionnement et du système de fichiers compris par la ROM.

ALTERNATIVE **Amorce tertiaire**

Certains systèmes se paient le luxe d'une amorce tertiaire. C'est le cas de FreeBSD, où on la trouve dans /boot/loader. Ce fichier est un maillon indispensable supplémentaire dans la chaîne de démarrage de la machine.
L'amorce tertiaire de FreeBSD est un programme très complet, qui reproduit un environnement semblable à l'OpenFirmware des PowerMacintosh et des stations de travail Sun et IBM. On peut l'utiliser pour spécifier des options de démarrage particulières, telles que la désactivation de la DMA sur un disque IDE, ou l'affichage d'un écran de démarrage. Elle permet aussi de choisir quels modules doivent être chargés dans le noyau au démarrage. /boot/loader se configure via le fichier /boot/loader.conf.

Démarrage du noyau

Une fois invoqué, le noyau procède à l'initialisation de la machine : configuration du processeur pour le placer dans les modes adéquats pour la gestion de la mémoire et choisir entre gros boutisme (*big endianness*) ou petit boutisme (*little endianness*), blocage des interruptions...

Vient ensuite la découverte du matériel de la machine. Chaque pilote présent dans le noyau utilise une sonde pour contrôler la présence dans la machine de matériel qu'il est capable de gérer. Quand le pilote gère plusieurs variantes du même matériel, cette sonde est aussi le moyen de les identifier clairement. En cas de découverte d'un périphérique qu'il connaît, le pilote l'initialise, le noyau enregistre l'association entre le périphérique et le pilote, et on passe au pilote suivant. Dans le cas contraire, le pilote ne sera pas utilisé, sauf introduction ultérieure d'un périphérique qu'on peut brancher « à chaud » (*hot plug*), comme une carte PCMCIA ou un périphérique USB.

Cette étape de reconnaissance du matériel peut donner lieu à des plantages sur les machines équipées de périphériques qu'on ne peut sonder de manière fiable. Ce problème, concernant principalement les PC à base de bus ISA, a été décrit dans le chapitre 4. Autre cas de plantage possible : quand un périphérique souffre d'un problème matériel et que son sondage plante la machine. Pour éviter ces désagréments, certains Unix permettent d'intervenir avant la phase de détection du matériel. Sur les systèmes BSD, ceci se fait en démarrant le noyau avec l'option **`-c`**. Sous NetBSD, par exemple, pour démarrer le noyau du premier disque SCSI dans ce mode, il faut interrompre l'amorce lorsqu'elle s'apprête à invoquer le noyau, et demander le démarrage sur **`sd0:/netbsd -c`** (adaptez les noms du disque et du noyau à votre configuration).

EN PRATIQUE **Nom des disques dans l'amorce sur PC**

Sur PC, l'amorce doit utiliser les procédures du BIOS pour accéder aux disques. Ils sont donc numérotés suivant la vision du BIOS, qui peut être différente de celle du noyau une fois qu'il a démarré.
Pour éviter les confusions, sous NetBSD et OpenBSD, l'amorce désigne les disques par `hd` au lieu de `wd` ou `sd`. Ainsi, pour démarrer sur le premier disque au sens du BIOS, on tapera : **`hd0:/netbsd`**.
FreeBSD a une approche différente ; on y indique le nom du disque pour FreeBSD en plus d'un nom destiné au BIOS. Exemple : **`0:ad(0,a)/kernel`** (disque **`0`** pour le BIOS, disque **`ad0a`** pour FreeBSD).

Choix de la racine

Quand le noyau a détecté le matériel, il monte la racine du système de fichiers. Il a pour cela besoin de deux informations, configurées lors de sa compilation : l'emplacement de cette racine, et le type de système de fichiers. Le noyau essaie automatiquement les réglages par défaut qu'on lui a indiqués. Si cela ne fonctionne pas, certains noyaux plantent, mais d'autres interrompent le démarrage automatique et s'enquièrent de ces renseignements sur la console. C'est le cas des systèmes BSD. On peut aussi indiquer l'option de démarrage **`-a`** pour forcer le

noyau à poser la question. Dans l'exemple de la section précédente, cela s'écrit : **`sd0:/netbsd -a`**

Remarquons que souvent, les noyaux Unix sont incapables d'utiliser certains systèmes de fichiers pour la racine. Pour les systèmes BSD, les classiques éprouvés sont FFS, ISO 9660, et les montages via le réseau en NFS ou par un disque virtuel intégré au binaire du noyau.

Pour les démarrages via le réseau, le noyau doit trouver son adresse IP et l'adresse du serveur NFS hébergeant la racine via DHCP ou BootP. Il faut ensuite monter cette racine. Pourquoi le noyau doit-il donc retrouver une adresse IP via DHCP, alors que la ROM ou l'amorce l'a déjà fait ? Car il n'existe aucun mécanisme pour transmettre au noyau ces paramètres ; ils sont donc oubliés lors du démarrage du noyau, qui doit les redécouvrir.

B.A.-BA **Partition racine**

La partition racine (on parle souvent de « racine » par abus de langage) est celle qui contient la racine du système de fichiers. Elle est montée par le noyau. Les autres partitions seront montées sur les points de montage adéquats par les scripts de démarrage. La racine correspond au répertoire / (*slash*).

CULTURE **Systèmes de fichiers**

FFS est l'acronyme de *Fast FileSystem* (système de fichiers rapide). Développé à Berkeley pour le BSD initial, il est encore utilisé par les trois systèmes BSD libres. FFS est une évolution de *Unix FileSystem* (UFS), utilisé par certains UNIX System V.
Certains Unix utilisent des systèmes de fichiers maison : ext2fs puis ext3fs sous Linux, HFS+ sur MacOS X, XFS sous IRIX, AdvFS sous Digital UNIX...

SUR LES AUTRES UNIX **`init` et `mach_init`**

Les systèmes basés sur le micro-noyau Mach (comme NeXTStep et MacOS X) sont une exception : c'est le programme **`mach_init`** qui y est le premier processus invoqué. Il crée rapidement un premier processus fils, qui garde le nom **`mach_init`**, tandis que lui-même devient le processus **`init`** traditionnel des systèmes Unix. Ceci donne l'impression que c'est le premier processus démarré (il a le PID 1 et **`mach_init`**, le PID 2), mais c'est **`mach_init`** qui est exécuté en premier. Le rôle de **`mach_init`** est d'assurer un service de noms pour les communications inter-processus du micro-noyau Mach.

SUR LES AUTRES UNIX **GNU/Linux**

Sous GNU/Linux, **`init`** demande le mot de passe de root pour accéder au mode mono-utilisateur. Pour contourner ce problème, il suffit de dire au noyau de démarrer /bin/bash à la place de /sbin/init, ce qui se réalise comme suit : **`boot: linux init=/bin/bash`**.

Mode mono-utilisateur

Une fois la racine montée, le noyau recherche le programme **`init`** pour l'exécuter. **`init`** a pour tâche de démarrer tous les autres processus de la machine. Avant son intervention, la machine est en mode mono-utilisateur, également appelé mode maintenance. **`init`** procède à l'initialisation du système, d'une façon assez différente suivant que l'on soit sur un Unix BSD ou System V, comme nous le verrons plus loin. Une fois l'initialisation terminée, on passe en mode multi-utilisateurs, mode de fonctionnement normal de la machine.

Si l'initialisation du système se passe mal, **`init`** reste en mode mono-utilisateur et démarre un shell avec les droits de root sur la console de la machine. On peut éventuellement configurer la machine pour demander le mot de passe de root avant de donner accès à ce shell, mais cette disposition peut être facilement contournée en indiquant au noyau d'utiliser /bin/sh comme programme d'initialisation, à la place d'**`init`**.

On peut aussi vouloir volontairement rester en mode mono-utilisateur. C'est ce que l'on fait lorsque la machine plante pendant l'initialisation du système, ou plus communément lorsqu'elle démarre mais qu'on arrive pas à s'y connecter après passage en mode multi-utilisateurs – si par exemple on a oublié le mot de passe de root. Le démarrage en mode mono-utilisateur s'obtient en indiquant au noyau une option de démarrage, qui sur les BSD et beaucoup d'autres systèmes est **`-s`**.

Les embûches du mode mono-utilisateur sont nombreuses : clavier mal configuré, variables d'environnement non définies, répertoires non accessibles en écriture... Voyons comment désamorcer ces pièges.

Configuration du clavier

Il est possible que le clavier de la console n'ait pas été configuré et qu'il faille taper en **`QWERTY`** américain sur un clavier **`AZERTY`** français. C'est désagréable, mais bénin : il suffit de taper la commande adéquate pour configurer la console

en clavier français. Malheureusement, cette commande varie très fortement d'un système à l'autre. Apprenez-la pour les systèmes que vous manipulez, apprenez le clavier américain par cœur, ou n'utilisez que des claviers **QWERTY**. Pour revenir à un clavier français sous NetBSD et OpenBSD, il faut taper : **/sbin/wsconsctl -w encoding=fr**, commande qui a le mauvais goût d'utiliser des caractères situés à des positions différentes entre ces deux claviers (sur un clavier français, on tapera donc **!sbin!zsconsctl -z encoding=fr**).

Pour FreeBSD, tapez : **/usr/sbin/kbdcontrol -l fr.iso.acc** (qui devient **!usr!sbin!kbdcontrol -l fr:iso:qcc** sur un clavier français).

Sous NetBSD et OpenBSD, le noyau doit contenir la « table de touches » concernée, ce qui est acquis sur i386, mais ce qui peut s'avérer moins évident sur des plateformes plus exotiques, personne n'ayant mis en place la table du clavier français. Sous FreeBSD, la commande **kbdcontrol** donnée ci-dessus fonctionnera en présence du fichier /usr/share/syscons/keymaps/fr.iso.acc.kbd, ce qui peut être difficile si cette partie de l'arborescence n'a pas encore été montée. Si /usr n'est pas monté, vous n'aurez de toutes façons pas accès à **kbdcontrol**...

Écrire sur le disque

Au démarrage de la machine, la racine est montée en lecture seule. Lors de l'invocation d'un shell en mode mono-utilisateur, elle s'y trouve souvent encore. Pour pouvoir écrire sur le disque, il faudra la remonter en lecture/écriture. La commande concernée varie légèrement d'un Unix à l'autre, et on passe parfois des mauvais moments à essayer de se souvenir de sa syntaxe, alors que de nombreuses personnes attendent impatiemment que le système redémarre. Exemple pour les BSD : **mount -o rw /dev/sd0a /**, en supposant évidemment que /dev/sd0a désigne la partition contenant la racine. Si vous ne connaissez pas cette partition, la commande **df** vous l'indiquera. Sur les BSD, on peut encore utiliser la commande **mount -o rw -u /**, qui fonctionnera même si l'on ne connaît pas le nom de la racine. Consultez la page **man** de **mount** pour en savoir plus.

En cas d'arrêt brutal de la machine, il faut songer à contrôler et réparer le système de fichiers avant de le monter en lecture/écriture. Ceci se fait avec la commande **fsck**, comme ceci : **fsck -y /dev/sd0a**

Faire l'impasse sur cette étape peut vous mener à un plantage ou à des pertes de données en cas de gros problèmes sur le système de fichiers ; c'est pourquoi il est toujours préférable d'utiliser **fsck** avant de remonter la racine en lecture/écriture. À l'inverse, dans certaines situations, la machine plantera pendant le **fsck**. Dans ce cas, vous êtes probablement mûr pour la reformatage de la racine, voire pour le remplacement du disque. Nous verrons comment gérer cette situation de crise à la fin du présent chapitre.

S'il faut monter d'autres partitions, procédez de même : **fsck** puis montage. Si le fichier /etc/fstab contient une entrée valide pour la partition, vous pouvez vous

PLUS LOIN **devfs**

Certains Unix (et en particulier quelques distributions Linux récentes et FreeBSD à partir de la version 5.0) utilisent un pseudo-système de fichiers baptisé devfs. Monté sur /dev, son rôle consiste à faire apparaître dynamiquement les fichiers spéciaux correspondant au matériel pris en charge. Ceci inclut les fichiers spéciaux donnant accès au disque – qui sont donc inaccessibles lorsque le devfs n'est pas en fonction.
Sans devfs, impossible donc de remonter la racine en lecture/écriture. Sous FreeBSD, **init** se charge de monter le devfs, il n'y a donc pas de problème. Sous GNU/Linux, le montage se fait en invoquant le *daemon* **devfsd**.

ATTENTION **Option -y dangereuse**

En temps normal, **fsck** est une commande interactive, qui demande à l'utilisateur ce qu'il faut faire face à chacun des problèmes qu'elle rencontre. L'option **-y** pousse **fsck** à travailler automatiquement, comme si vous répondiez positivement à chacune des questions qu'elle pose. C'est donc une option potentiellement dangereuse, qui peut vous amener à faire des choses que vous ne souhaitez pas.
Cependant, faire fonctionner **fsck** sans l'option **-y** peut obliger à répondre à des centaines de questions, dont les tenants et les aboutissants peuvent être difficiles à saisir. Comme la plupart du temps **fsck -y** fait du très bon travail, on se lance rarement dans un **fsck** interactif.

SUR LES AUTRES UNIX **GNU/Linux**

Sous GNU/Linux, la commande de remontage de la racine en lecture/écriture est un peu difficile à retrouver lors d'un besoin urgent. Tout se trouve dans les pages **man**, mais en voilà le résumé : **mount -n -o rw,remount /dev/sda1 /**

contenter d'indiquer le point de montage (par exemple /usr) à **fsck** ou **mount**, sinon il faut indiquer le fichier périphérique concerné (comme /dev/sd0e).

Problèmes de variables d'environnement

Premier problème : le PATH, qui contient la liste des répertoires où trouver les commandes tapées, peut être indéfini ou incomplet. Cela se règle simplement, en **sh** :

```
# PATH=/bin:/sbin:/usr/bin:/usr/sbin
# export PATH
```

En **ksh** ou **bash**, on peut résumer cela en une ligne :

```
# export PATH=/bin:/sbin:/usr/bin:/usr/sbin
```

En **csh** ou **tcsh** cela devient :

```
# setenv PATH /bin:/sbin:/usr/bin:/usr/sbin
```

Autre problème courant : la variable TERM, qui indique le type du terminal, n'est pas initialisée. Ceci posera des problèmes à tout programme fonctionnant en pleine page, comme par exemple **vi**. À défaut de connaître précisément le type du terminal, indiquer **vt100** permet souvent de fonctionner, même dans un mode légèrement dégradé.

PLUS LOIN **Terminal générique**

Le réglage **vt100** fonctionne bien dans la plupart des cas, mais il est indispensable de configurer le bon terminal sur certaines machines. On indiquera par exemple **sun** sur les stations Sun.

Édition des lignes, rappel des commandes, complétion

Même en mode mono-utilisateur, on aime son petit confort. Pour disposer de l'édition des lignes, du rappel des commandes et de la complétion, il suffit d'invoquer un shell capable d'assurer ces fonctions, et de le configurer correctement. Exemple avec **ksh**, présent presque partout, et notamment sur NetBSD et OpenBSD :

```
# ksh
# set -o emacs
```

Si le terminal est bien configuré, vous pourrez éditer les lignes et revenir dans l'historique avec les touches fléchées. Dans le cas contraire, il faudra utiliser des combinaisons de touches telles que **Control**+**P** (monter dans l'historique), **Control**+**N** (descendre dans l'historique), **Control**+**B** (se déplacer à gauche), **Control**+**F** (se déplacer à droite), **Control**+**A** (se rendre en début de ligne), et **Control**+**E** (se rendre en fin de ligne). La complétion se fera alors en enfonçant deux fois la touche **Échap** (ou **Control**+**[** en son absence).

SUR LES AUTRES UNIX **Shells par défaut sous FreeBSD et GNU/Linux**

FreeBSD et la plupart des distributions GNU/-Linux n'incluent pas **ksh** dans leur distribution de base, mais en ce qui concerne le confort, cela ne pose pas de problème : FreeBSD fournit en standard **tcsh**, et **bash** est le shell par défaut sur pratiquement toutes les distributions GNU/-Linux. Ces deux shells savent gérer automatiquement l'édition de la ligne, la complétion et le rappel des commandes passées.

ASTUCE **Complétion**

Si vous utilisez un Korn Shell récent (c'est le cas sur les BSD), vous pouvez configurer la complétion sur la touche **Tabulation** : **bind '^I'=complete-file**. En cas d'absence ou de dysfonctionnement de cette touche, on pourra la simuler avec la combinaison de touches **Control**+**I**.

Utiliser `vi`

vi acceptera de fonctionner si TERM correspond à un terminal décrit dans la base de terminaux, écrite au format termcap ou terminfo, suivant le système. Les BSD préfèrent termcap, qui se trouve sous NetBSD et FreeBSD dans le fichier /usr/share/misc/termcap.db. Les System V et OpenBSD préfèrent terminfo, situé dans /usr/share/misc/terminfo.db. Dans ces deux cas, il s'agit d'un fichier indexé en binaire, pour un accès plus rapide. En l'absence de ces fichiers ou si TERM a une valeur non conforme, **vi** ne fonctionnera pas.

Deuxième exigence de **vi** : les répertoires /var/tmp et /tmp doivent être accessibles en écriture. Si cela est impossible (partition hors service, ou démarrage sur un média en lecture seule tel qu'un CD-ROM), on peut monter des disques virtuels sur ces points de montage. Sur NetBSD et OpenBSD, cela peut se faire comme suit, si le noyau est capable de gérer des disques virtuels :

```
# mount -t mfs swap /var/tmp
# mount -t mfs swap /tmp
```

Les répertoires concernés deviennent alors accessibles en écriture, mais ce qu'on y inscrit est stocké dans la mémoire vive de la machine : on perdra donc ces données au redémarrage, et la place est limitée.

EN CAS DE COUP DUR **`vi` n'existe plus ?**

Cas critique où **vi** refuse de fonctionner : /usr est sur une partition séparée de la racine. Lorsque /usr n'est pas monté, **vi** est évidemment inaccessible. La solution est simple : après un éventuel **fsck**, montez /usr en tapant **mount /usr**. Vous pouvez aussi utiliser **ed**.

Passage en mode multi-utilisateurs

Le passage en mode multi-utilisateurs diffère beaucoup entre les systèmes BSD et les UNIX System V. Pour compliquer le tout, NetBSD puis FreeBSD ont récemment adopté un système hybride : l'initialisation **rc.d-ng**.

Initialisation des systèmes BSD

Le passage en mode multi-utilisateurs sur un système BSD classique est très simple : **init** invoque un shell qui exécute le script /etc/rc. Ce dernier lit la configuration du système dans le fichier /etc/rc.conf, puis démarre tous les démons (*daemons*) qui y sont précisés.

/etc/rc exécute enfin /etc/rc.local, qui permet à l'ingénieur système de préciser certaines tâches à exécuter au démarrage, sans devoir modifier /etc/rc. À la fin de son exécution, la machine est en mode multi-utilisateurs. La seule tâche qui incombe alors à **init** est d'invoquer des processus **getty** sur les terminaux listés comme actifs dans le fichier /etc/ttys. Le rôle de ces processus est d'initialiser les terminaux et de présenter une invite de connexion (*login*). Voici quelques lignes d'un fichier /etc/ttys :

```
ttyE0   "/usr/libexec/getty std.9600"   vt100   on secure
tty00   "/usr/libexec/getty std.38400"  vt100   on secure
tty01   "/usr/libexec/getty std.9600"   unknown off secure
tty02   "/usr/libexec/getty std.9600"   unknown off secure
```

VOCABULAIRE **`rc.d-ng`**

Le terme vient du répertoire rc.d, où sont stockés les scripts d'initialisation à la System V, et d'un acronyme signifiant « nouvelle génération ».

EN PRATIQUE **Variables pour rc.conf**

Sur NetBSD et FreeBSD, vous pouvez trouver dans /etc/defaults/rc.conf toutes les options utilisables dans /etc/rc.conf.

CULTURE **Les daemons**

Quoi qu'en pensent les utilisateurs débutants, Unix n'est pas une antichambre de l'enfer, et les démons qui l'habitent ne sont pas des créatures cornues. Le mot *daemon* n'est qu'un acronyme (*Disk And Extension MONitor* : moniteur de disque et d'extension – traduit en français par « démon » par abus de langage). Il désigne tout programme exécuté en tâche de fond et n'interagissant pas avec l'utilisateur par le biais d'un terminal.

SUR LES AUTRES UNIX **Scripts de démarrage sous MacOS X**

À l'instar de son ancêtre NeXTStep, MacOS X n'est pas un BSD comme les autres. **init** y exécute d'abord le script /etc/rc.boot, qui en cas de démarrage à travers le réseau invoque /etc/rc.netboot. /etc/rc n'est ensuite invoqué que pour le passage en mode multi-utilisateurs.

PLUS LOIN **init et getty**

C'est **getty** qui propose l'invite de connexion présentée sur chaque terminal de la machine. On peut ajouter des terminaux virtuels sur la console, accessibles avec les combinaisons de touches **Control**+**Alt**+**F1** à **Control**+**Alt**+**F12**. Pour cela, il faut d'abord les ajouter au niveau du pilote de la console, grâce à **wsconscfg** sur NetBSD et OpenBSD, et **vidcontrol** sur FreeBSD. Pour configurer automatiquement cela au démarrage sous NetBSD, il faut préciser **wscons=YES** dans /etc/rc.conf et modifier le fichier /etc/wscons.conf. Pour OpenBSD, il faut modifier /etc/wsconsctl.conf. FreeBSD active par défaut 16 terminaux virtuels, ce qui devrait suffire.
Une fois le terminal créé, on peut demander au processus **init** d'y démarrer un **getty** en modifiant le fichier /etc/ttys et en lui envoyant un signal pour le prévenir de la modification, comme suit : **kill -1 1**

SUR LES AUTRES UNIX **BSD et niveaux d'exécution**

Les BSD ont des niveaux de sécurité, parfois assimilés à tort aux niveaux d'exécution d'**init** sur System V.
Les niveaux de sécurité des BSD ne concernent que le noyau. Ils autorisent ou interdisent certaines actions, comme le chargement de modules dynamiques ou l'accès au matériel. Au moment du démarrage, le niveau de sécurité est réglé à 0, puis passe à 1 ou 2 en mode multi-utilisateurs. La commande **sysctl** permet de changer le niveau de sécurité du noyau.

On indique d'abord le nom du fichier spécial du répertoire /dev pour le terminal concerné, puis la commande à exécuter. Suivent le type de terminal, un `on` ou un `off` indiquant respectivement que **init** doit invoquer ou non un processus **getty** sur ce terminal, puis le mot-clef `secure`, facultatif. Les terminaux marqués `secure` acceptent une connexion de root, et les autres le refusent. De plus, sur toute console non marquée `secure`, **init** demandera le mot de passe de root avant de donner accès à un shell mono-utilisateur.

Tout cela paraît assez simple, sauf le jour où il faut lire le fichier /etc/rc. Parmi les BSD, seul OpenBSD démarre encore selon cette séquence classique. Avant d'examiner le cas de NetBSD et de FreeBSD, voyons le démarrage des UNIX System V.

Initialisation des UNIX System V

Les UNIX System V ont un démarrage beaucoup plus complexe. **init** y dispose de plusieurs niveaux de fonctionnement, ou d'exécution (*runlevels* en anglais). Le mode mono-utilisateur est le niveau 1, et on en trouve 6 autres : 0 sert à l'arrêt de la machine, 6 à son redémarrage, et les niveaux 2 à 5 aux modes multi-utilisateurs, car il y en a plusieurs. Au démarrage, **init** lit le fichier /etc/inittab et y cherche l'entrée `initdefault`, qui lui indique son niveau d'exécution par défaut. C'est aussi /etc/inittab qui indique à **init** quoi faire dans chaque niveau. Ce sont par exemple les lignes suivantes qui invoquent les processus **getty** :

```
S0:2345:respawn:/sbin/getty 9600 ttyS0
```

Chaque ligne contient des champs séparés par le caractère **:** (deux points). Le premier champ indique un nom pour la ligne. `2345` précise ensuite que cette commande concerne les niveaux 2, 3, 4, et 5, et `respawn` spécifie que les processus concernés doivent être redémarrés dès qu'ils meurent. Le dernier champ indique la commande à exécuter.

On trouve de plus des entrées :

```
l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
```

qui indiquent d'invoquer le script /etc/init.d/rc en lui passant un niveau d'exécution. Ce script exploitera par exemple les éléments du répertoire /etc/rc3.d si on lui passe l'argument **3**. Ce répertoire contient des liens vers des scripts du répertoire /etc/init.d. Exemple de répertoire /etc/rc3.d pour une distribution Debian GNU/Linux :

```
# ls -l
total 0
lrwxrwxrwx  1 root   root    18 Apr 12  2001 S10sysklogd -> ../init.d/sysklogd
lrwxrwxrwx  1 root   root    15 Apr 13  2001 S11klogd -> ../init.d/klogd
lrwxrwxrwx  1 root   root    15 Apr 12  2001 S20inetd -> ../init.d/inetd
lrwxrwxrwx  1 root   root    17 Apr 12  2001 S20logoutd -> ../init.d/logoutd
```

```
lrwxrwxrwx  1 root   root    17 Jan 20  2002 S20makedev -> ../init.d/makedev
lrwxrwxrwx  1 root   root    17 Apr 13  2001 S20postfix -> ../init.d/postfix
lrwxrwxrwx  1 root   root    13 Apr 12  2001 S20ssh -> ../init.d/ssh
lrwxrwxrwx  1 root   root    17 Apr 13  2001 S22ntpdate -> ../init.d/ntpdate
lrwxrwxrwx  1 root   root    13 May 15  2001 S23ntp -> ../init.d/ntp
lrwxrwxrwx  1 root   root    14 Apr 13  2001 S89cron -> ../init.d/cron
lrwxrwxrwx  1 root   root    19 Apr 12  2001 S99rmnologin -> ../init.d/rmnologin
```

Les scripts sont invoqués un par un, par ordre lexicographique. On joue donc sur le numéro situé au début du nom pour modifier l'ordre d'exécution. Suivant que la première lettre du lien soit `K` (*Kill*, ou tuer) ou `S` (*Start*, ou démarrer), les scripts seront invoqués avec l'argument **`stop`** ou **`start`**, respectivement.

Invoquer un script du répertoire /etc/init.d avec l'argument **`start`** permet évidemment de démarrer le *daemon* correspondant, alors que l'argument **`stop`** lui enjoindra de s'arrêter. Ces scripts peuvent d'ailleurs être utilisés manuellement pour arrêter ou démarrer des services. On peut aussi les appeler avec l'argument **`restart`** pour redémarrer un service donné.

Tout cela peut paraître excessivement compliqué, et c'est probablement le cas. Mais l'initialisation à la System V a quand même un gros avantage sur l'initialisation à la BSD : toute modification d'un script de /etc/init.d peut immédiatement être validée en invoquant ce dernier avec l'argument **`start`** ou **`restart`**. On risque donc peu de surprises au redémarrage.

Sur un système BSD, on configure l'initialisation de la machine en modifiant /etc/rc.conf et /etc/rc.local. Cette méthode a l'avantage d'être simple, mais elle a un inconvénient : le seul moyen de contrôler les fichiers de configuration /etc/rc.conf et /etc/rc.local est de redémarrer la machine. Les systèmes BSD étant très stables, un redémarrage non exécuté volontairement peut mettre longtemps à se produire. Une coquille dans /etc/rc.local n'est donc parfois mise en évidence que plusieurs mois plus tard, après éventuel oubli de la modification effectuée.

NetBSD a donc mis au point un système hybride, tentant de concilier le meilleur des deux mondes : **`rc.d-ng`** – système récemment adopté par FreeBSD.

ATTENTION **À propos du fichier inittab**

Comme pour le fichier /etc/ttys des BSD, il faut signaler à **`init`** la modification du fichier /etc/inittab en lui envoyant un signal, à l'aide de la commande **`kill -1 1`**

PLUS LOIN **Revenir en mode mono-utilisateur**

Une fois passé en mode multi-utilisateurs, on peut souhaiter redescendre en mode mono-utilisateur sans forcément redémarrer, pour mettre en place par exemple une mise à jour de sécurité.
Sur les systèmes BSD, la commande **`kill -1 1`** permet de retrouver le mode mono-utilisateur. Elle provoquera l'arrêt de tous les processus à l'exception de **`init`**, et un shell avec les droits de root sera invoqué sur la console de la machine.
Sur UNIX System V et sous GNU/Linux, on obtiendra le même résultat en tapant **`telinit 1`**.

SUR LES AUTRES UNIX **Démarrage des services sous MacOS X**

MacOS X se distingue encore sur l'étape de démarrage des services. La commande **`SystemStarter`** invoque tous les services décrits dans les sous-répertoires de /System/Library/StartupItems et /Library/StartupItems.
Chaque service y est représenté par un script d'initialisation à la System V, qui porte le même nom que le répertoire, et par un fichier StartupParameters.plist, qui sert à l'ordonnancement du démarrage. Exemple d'un tel fichier :

```
{
  Description     = "secure login server";
  Provides        = ("SSH");
  Requires        = ("Resolver");
  OrderPreference = "None";
  Messages =
  {
    start = "Starting secure login server";
    stop  = "Stopping secure login server";
  };
}
```

Initialisation BSD nouvelle génération

NetBSD et FreeBSD restent des systèmes BSD : on n'y trouve pas de niveaux d'exécution ni de fichier /etc/inittab. **`init`** commence par exécuter /etc/rc, qui lit toujours sa configuration dans /etc/rc.conf. La nouveauté, c'est que /etc/rc n'est plus un script monstrueux chargé d'initialiser tout le système. Il va exécuter des scripts situés dans /etc/rc.d, chacun étant chargé du démarrage d'un *daemon*.

Chaque script s'invoque avec l'argument **`start`**, **`stop`**, ou **`restart`**, à l'image des scripts d'initialisation d'UNIX System V. La similitude n'est toutefois pas totale : pour éviter le bricolage consistant à faire des liens avec des noms commençant par `K` ou `S` et contenant des numéros, tous les scripts sont exécutés au démarrage avec l'argument **`start`**. Chaque script de /etc/rc.d vérifie si les directives présentes dans le fichier /etc/rc.conf lui demandent de réellement exécuter le *daemon* dont il a la charge.

L'ordonnancement est géré différemment : pour connaître l'ordre de démarrage des scripts, /etc/rc fait appel au programme **`rcorder`**, qui lit les premières lignes de chaque script et y trouve des commentaires contenant les mots-clef `PROVIDE` (fournir), `BEFORE` (avant), et `REQUIRE` (requiert). Ces codes lui permettent d'établir l'ordre de démarrage. Donnons l'exemple de /etc/rc.d/nfsd :

```
#!/bin/sh
#
# $NetBSD: nfsd,v 1.4 2001/06/16 06:13:10 lukem Exp $
#

# PROVIDE: nfsd
# REQUIRE: mountd
```

`rcorder` sait ainsi que le script nfsd fournit le service **`nfsd`**, mais ne doit pas être exécuté tant que le service **`mountd`** n'a pas démarré.

L'initialisation rc.d-ng est plus complexe que l'initialisation historique des BSD, mais permet de tester si un service démarre correctement sans devoir redémarrer la machine. Il suffit d'exécuter le script correspondant dans /etc/rc.d avec l'argument **`start`**. Si cela fonctionne, cela fonctionnera encore au prochain démarrage de la machine. Et qui est rebuté par ce style rappelant System V pourra toujours continuer à utiliser NetBSD et FreeBSD comme des systèmes à initialisation BSD pure, en ignorant /etc/rc.d et en ne modifiant que les fichiers /etc/rc.conf et /etc/rc.local.

PIÈGE À C… **Chemin des scripts**

Pour invoquer un service depuis /etc/rc.d, il faut penser au préfixe ./. En effet, la variable `$PATH` ne le contient pas toujours (c'est même déconseillé, car potentiellement dangereux), et taper **`sshd start`**, exécutera vraisemblablement /usr/sbin/sshd au lieu de /etc/rc.d/sshd, ce qui n'aura donc pas l'effet escompté.

Systèmes dynamiques

Sur presque tous les systèmes d'exploitation, les programmes sont liés avec des bibliothèques (*libraries*) lors de leur compilation. Ces dernières renferment du code commun à plusieurs programmes ; leur rôle est d'éviter que les développeurs passent leur temps à reprogrammer sans cesse les mêmes fonctionnalités. On dispose par exemple de bibliothèques de fonctions d'affichage dans un terminal, de calcul numérique, d'affichage de fenêtres ou de menus, et bien d'autres encore.

VOCABULAIRE **Bibliothèques ou librairies ?**

Comme en anglais, langue originelle et véhiculaire du domaine, on parle de *library*, beaucoup d'informaticiens français ont pris l'habitude d'utiliser le contresens « librairie » pour évoquer ce concept, en raison de ses sonorités proches.

Les bibliothèques peuvent être incluses dans chaque programme ; on parle alors de programme statique. Le programme peut aussi se contenter de faire référence aux fonctions des bibliothèques, et ne sera relié aux bibliothèques dont il a besoin qu'au moment de l'exécution. On parle alors de programme et de bibliothèques dynamiques.

Sur la plupart des systèmes Unix, dont les systèmes BSD, la commande **`file`** indique la nature d'un fichier en testant son contenu (les noms et extensions de fichiers étant libres). Elle permet notamment de savoir si un programme est statique ou dynamique. Dans le cas d'un programme dynamique, la commande **`ldd`** indiquera la liste des bibliothèques dont il dépend :

```
$ file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (FreeBSD), statically
linked, stripped

$ file /usr/bin/vi
/usr/bin/vi: ELF 32-bit LSB executable, Intel 80386, version 1 (FreeBSD), dynami
cally linked (uses shared libs), stripped

$ ldd /usr/bin/vi
/usr/bin/vi:
        libncurses.so.5 => /usr/lib/libncurses.so.5 (0x280a8000)
        libc.so.4 => /usr/lib/libc.so.4 (0x280ea000)
```

La liaison dynamique a des avantages et des inconvénients. Elle permet d'économiser de la place, tant sur le disque qu'en mémoire, puisqu'une bibliothèque n'est chargée qu'une fois, puis partagée entre les différents programmes qui l'utilisent. De plus, il suffit de mettre à jour une seule bibliothèque partagée pour mettre à jour tous les programmes dynamiques qui en dépendent, alors que dans un tel cas de figure il faut remplacer tous les programmes statiques.

Malheureusement, les performances sont sacrifiées : la liaison dynamique prend du temps, là où un programme statique peut s'exécuter tout de suite. Le système se trouve de plus fragilisé : qu'une seule bibliothèque soit endommagée ou supprimée, et de nombreux programmes peuvent être rendus inopérants.

Tout ceci influe sur la procédure de démarrage. Sur certains systèmes, les programmes critiques tels que /bin/sh ou /sbin/init sont liés statiquement. C'est le cas pour FreeBSD, OpenBSD, et pour NetBSD avant la version 2.0. D'autres systèmes sont entièrement dynamiques : tous les programmes, sans exception, y sont dynamiques. C'est le cas de GNU/Linux, des UNIX System V, de MacOS X, et de NetBSD à partir de la version 2.0.

L'impact sur la séquence de démarrage, c'est l'apparition de nouveaux points faibles. Si la liaison dynamique du programme **`init`** échoue, le système plantera au démarrage. Il est donc bon de savoir quels sont les fichiers indispensables à la liaison dynamique si l'on souhaite pouvoir remettre sur pieds un système en perdition.

Il faut disposer de :

- Toutes les bibliothèques dont dépend le programme ;
- et toutes les bibliothèque dont dépendent les bibliothèques précédentes, et ainsi de suite (MacOS X est le champion toutes catégories des dépendances

CULTURE **Bibliothèques dynamiques**

Sous Unix, les bibliothèques dynamiques sont souvent des fichiers .so (*shared object*, ou objet partagé). Sur un système BSD, par exemple, /usr/lib/libm.so est une bibliothèque contenant des fonctions mathématiques.
Les bibliothèques statiques ont pour extension .a (archive). Elles ne servent qu'à la compilation des programmes, pas à leur exécution, à la différence des bibliothèques dynamiques, indispensables au fonctionnement de tout programme dynamique. Chaque bibliothèque statique dispose d'un pendant dynamique : /usr/lib/libm.a (statique) correspond par exemple à /usr/lib/libm.so (dynamique).

SUR LES AUTRES UNIX **MacOS X**

Sous MacOS X, les bibliothèques dynamiques portent l'extension .dylib (*dynamic library*). La commande **`ldd`** n'existe pas ; c'est **`otool -L`** qui joue son rôle :

```
$ otool -L /bin/ls
/bin/ls:
        /usr/lib/libSystem.B.dylib
(compatibility version 1.0.0,
current version 60.0.0)
```

emboîtées, mais on en trouve sur d'autres systèmes. **ldd** peut être utilisé sur une bibliothèque sur comme un programme).

- L'éditeur de liens dynamiques est le programme chargé de lier un programme dynamique à ses bibliothèques. Il est commun à tous les binaires de même format d'exécution (ELF, a.out, Mach-O...) et son nom dépend du système. Quelques exemples :

NetBSD, FreeBSD, OpenBSD a.out	/usr/libexec/ld.so
NetBSD (avant 2.0), OpenBSD ELF	/usr/libexec/ld.elf_so
NetBSD (à partir de 2.0) ELF	/libexec/ld.elf_so
FreeBSD ELF	/usr/libexec/ld-elf.so.1
Linux ELF ou a.out	/lib/ld.so.1
IRIX ELF o32	/lib/libc.so.1
IRIX ELF n32	/lib32/libc.so.1
MacOS X Mach-O	/usr/lib/dyld

- Sous UNIX System V (mais pas sous GNU/Linux), il faut disposer d'un /dev/zero fonctionnel.

NetBSD a su préserver la fiabilité dans son passage à un système entièrement dynamique : tous les binaires critiques et indispensables à la réparation du système sont présents en version statique dans le répertoire /rescue. De ce fait, dans le pire des cas, on pourra toujours démarrer le noyau avec les options **-s -a** pour lui préciser /rescue/init au lieu de /sbin/init, et démarrer en mode mono-utilisateur avec un **init** lié statiquement.

PLUS LOIN **/dev/zero**

Le fichier spécial /dev/zero est une source infinie de caractères nuls (de code ASCII zéro). Pour créer un gros fichier de dix millions de caractères entièrement constitué de zéros, on peut ainsi utiliser la commande suivante : **dd if=/dev/zero of=/tmp/test bs=1024 count=10000**.
/dev/zero a un fichier frère : le /dev/null, sorte de trou noir où tout ce qui est écrit est oublié. On y redirige des flux (de sortie ou d'erreur standard) dont on ne souhaite pas garder trace, ou on y branche une entrée standard quand une commande ne doit recevoir aucune donnée. La commande **cat /dev/zero > /dev/null** occupera donc le processeur à créer puis détruire des zéros, jusqu'à son interruption.

Restauration du système

Vous savez maintenant tout sur la séquence de démarrage, et avez les pistes pour traiter la plupart des problèmes qui peuvent survenir si la machine est encore capable de démarrer en mode mono-utilisateur.

Dans un scénario catastrophe, cela n'est plus possible. Il faudra alors sans doute démarrer sur un autre support que le disque habituel : disquette, CD-ROM, réseau, voire une partition système préparée sur un disque de la machine par l'administrateur prévoyant. On peut aussi transplanter le disque de la machine malade dans une machine dotée d'un disque en état de démarrer, et y procéder aux réparations.

Une fois la machine démarrée sur un système de secours, vous pourrez prendre les mesures adéquates :

- Réinstaller une amorce effacée par erreur.
- Mettre en place un noyau fonctionnel en cas d'installation d'un noyau qui plante au démarrage.
- Exécuter **fsck** sur la racine, si cette dernière est endommagée au point de ne pouvoir supporter cette opération après un démarrage normal.

ASTUCE **Les disquettes d'installation**

Les disquettes d'installation peuvent servir de système de secours pour démarrer. Elles donnent accès à un shell, et contiennent tous les programmes nécessaires à la remise en état du système.

- Reformater la racine, et restaurer son contenu depuis les sauvegardes effectuées régulièrement (car vous faites des sauvegardes régulières, bien entendu). On en arrive à cette extrémité si le système de fichiers est endommagé au point que **`fsck`** n'arrive pas à le réparer. Il faut ensuite penser à réinstaller l'amorce.
- Refaire le partitionnement du disque, reformater les partitions, tout restaurer depuis les sauvegardes, et réinstaller l'amorce, si par inadvertance tout a été cassé. Ne pensez pas que cela n'arrive jamais : tout véritable ingénieur système a forcément démoli (logiciellement) une machine par erreur, au moins une fois dans sa carrière.
- Même manipulation en changeant le disque, dans le cas où le disque de démarrage a rendu l'âme.

Il faut bien retenir qu'il n'y a pas de situation désespérée, et que tant que la machine n'est pas matériellement endommagée, on peut récupérer toutes les situations sans avoir à tout réinstaller et tout reconfigurer depuis zéro. Deux conditions à cela toutefois : disposer de sauvegardes et d'un moyen de démarrer une machine : disquette, CD-ROM, serveur de démarrage à travers le réseau, tous les coups sont permis. Mais il vaut mieux les avoir préparés et testés pour s'éviter des situations stressantes.

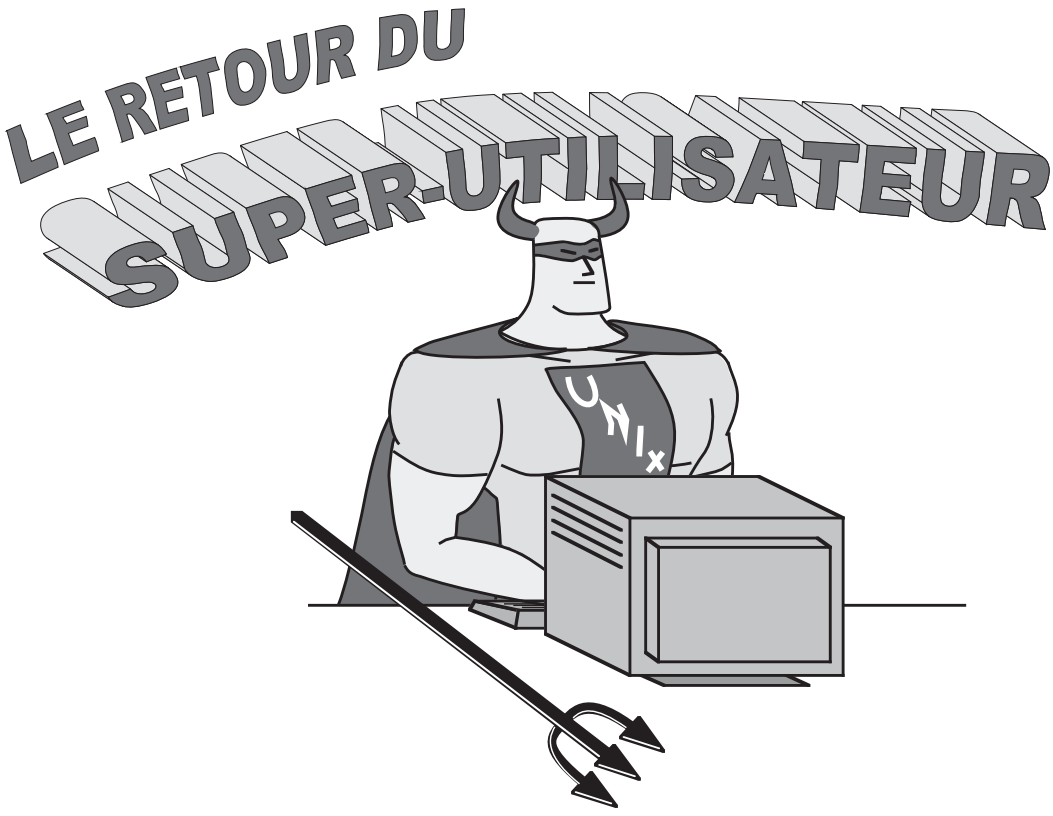
ATTENTION **Partitionnement, formatage, réinstallation d'amorce**

Ces opérations sont rarement mises en œuvre dans l'exploitation d'un serveur Unix, mais mieux vaut bien les connaître le jour où survient la panne, sous peine de manquer d'efficacité dans la gestion de l'incident.

L'utilisation des commandes **`fdisk`**, **`disklabel`**, **`newfs`**, **`installboot`** ou **`boot0cfg`** pour l'installation d'un système BSD est décrite au chapitre 4, dans la section consacrée à l'installation sans le programme d'installation. Ce qui y est décrit est également valable pour la maintenance, le but étant le même : obtenir un système capable de démarrer.

6

LE RETOUR DU SUPER-UTILISATEUR



Utilisateurs, groupes, et sécurité

C'est évidemment à l'administrateur que reviennent les tâches de gestion des utilisateurs et des groupes sur la machine – notions fondamentales sur un système Unix, dans la mesure où elles forment la base de ses mécanismes de sécurité.

SOMMAIRE

MOTS-CLEFS

- Utilisateurs et groupes
- Gestion des permissions
- Sécurité

La gestion des utilisateurs et groupes est typiquement le domaine où chaque Unix utilise des outils spécifiques. Leur raison d'être est de faciliter la vie de l'administrateur système, mais ils ont parfois le mauvais goût de la compliquer, en introduisant autant de comportements différents qu'il existe de versions d'Unix.

Heureusement, il est possible de gérer ses utilisateurs en s'en tenant aux outils et méthodes qui relèvent d'un tronc commun. C'est ce que nous allons examiner dans ce chapitre, qui se veut extrêmement générique.

Gérer ses utilisateurs

PIÈGE À C... **Tout faire en tant que root**
Certains administrateurs débutants s'interrogent parfois sur l'intérêt de mettre en place un compte utilisateur normal alors qu'ils peuvent travailler en permanence en tant que root.
La réponse à cette question s'imposera à eux – douloureusement – le jour où ils feront le ménage à la racine du disque en croyant se trouver sous /tmp, détruisant irrémédiablement tout le système.

Contrairement à ce que le titre pourrait laisser penser, nous n'aborderons pas ici l'aspect humain de la gestion des utilisateurs. Non que le sujet soit inintéressant, mais il pourrait faire l'objet d'un ouvrage entier. Nous nous cantonnerons donc à la gestion technique des comptes des utilisateurs. Une des premières tâches de l'administrateur d'un système Unix est de créer des comptes utilisateur. Même sur une machine sans utilisateur local, l'ingénieur système doit se créer un compte personnel.

Sur Unix canal historique

À l'origine, toutes les informations des comptes utilisateur se trouvaient dans le fichier /etc/passwd, y compris les mots de passe. Pour modifier des comptes, on intervenait sur ce fichier avec un éditeur de texte, tel que **`vi`**. Voici un exemple de fichier /etc/passwd :

```
root:2hKTuNumO4sU6:0:0:Charlie Root:/root:/bin/sh
daemon:*:1:31:The devil himself:/:/sbin/nologin
operator:*:2:5:System operator:/usr/guest/operator:/sbin/nologin
bin:*:3:7:Binaries Commands and Source:/:/sbin/nologin
news:*:6:8:Network News:/var/spool/news:/sbin/nologin
uucp:*:66:1:UNIX-to-UNIX Copy:/var/spool/uucppublic:/usr/libexec/uucp/uucico
manu:pCs71RItmHgdc:500:500:Emmanuel Dreyfus:/home/manu:/bin/ksh
```

Chaque ligne définit un compte utilisateur. Elle comprend sept champs séparés par le caractère `:` (deux points).

- L'identifiant de connexion (*login*) de l'utilisateur, limité à huit caractères sur certains systèmes. Même si ce n'est pas le cas de tous, il est conseillé de toujours s'en tenir à huit caractères maximum pour des raisons d'interopérabilité,
- Le mot de passe de l'utilisateur, chiffré avec un procédé à sens unique (*one-way function*) pour produire une valeur de hachage (*hash* en anglais) : c'est le mot de passe chiffré. On peut facilement chiffrer un mot de passe, mais il est difficile de retrouver un mot de passe produisant un hachage donné. Lorsque l'usager tente de se connecter au système, il fournit son mot de passe, qui est chiffré puis comparé à ce champ du fichier /etc/passwd. Si les versions chiffrées coïncident, le candidat est accepté. Dans le cas contraire, la connexion est refusée. Laissé vide, ce champ laisse un accès sans mot de passe. On peut encore y placer un `*` (astérisque) pour interdire toute connexion par mot de passe sur le compte en question.

- L'identifiant unique de l'utilisateur (UID), constamment utilisé par le système pour l'identifier. Ce nombre doit être compris entre 0 et (souvent) 32767, 0 étant réservé à l'administrateur. Certains systèmes permettent des UID supérieurs à 32767, mais pour des raisons d'interopérabilité, il est conseillé ici encore de s'en tenir au dénominateur commun.
- L'identifiant unique du groupe de l'utilisateur (GID), soumis aux mêmes limitations que l'UID. L'utilisateur peut appartenir à plusieurs groupes, comme nous le verrons plus loin. Dans ce cas, le numéro indiqué ici est celui de son groupe principal (on dit aussi « groupe primaire »).
- Le nom complet de l'utilisateur, éventuellement suivi de ses coordonnées : adresse, téléphone, etc.
- Le chemin du répertoire personnel de l'utilisateur. S'il n'existe pas, c'est la racine du système de fichiers qui sera utilisée comme répertoire par défaut au moment de la connexion. L'utilisateur n'y aura pas pour autant des droits particuliers...
- Le shell (interpréteur de commandes) par défaut de l'utilisateur. C'est le programme qui sera invoqué lors de la connexion. On peut ainsi empêcher quelqu'un de se connecter en session interactive en précisant ici un exécutable tel que /sbin/nologin. /sbin/nologin n'est pas un shell. Lorsqu'on l'invoque, il rend immédiatement la main, ce qui aura pour effet de terminer instantanément une tentative d'ouverture de session.

Le fichier /etc/passwd s'édite avec n'importe quel éditeur de texte, à l'exception de son champ de mot de passe, qu'on réserve souvent aux bons soins de la commande **passwd**. Invoquée par root, elle permet de changer le mot de passe de n'importe qui, sans devoir prouver au préalable sa connaissance de l'ancien mot de passe :

```
# passwd manu
Changing local password for manu.
New password:
```

EN PRATIQUE **Quels UID utiliser ?**

Les UID (numéros d'identifiant utilisateur) sont libres, à condition de ne pas entrer en conflit avec les UID réservés par le système. Sur BSD, les UID supérieurs à 200 ne sont pas alloués ; vous pouvez en faire ce qui vous plaît. Sur un certain nombre de systèmes Unix, l'UID maximum est 32767.

PIÈGE À C... **Le shell de root**

On peut trouver rudimentaire le shell par défaut de root (souvent, /bin/sh). Si vous désirez le changer, n'oubliez pas que root peut être amené à se connecter alors que /usr n'est pas encore monté. Préférez donc un shell lié statiquement, ou à défaut, un shell lié avec des bibliothèques présentes sur la partition racine du système de fichiers – le shell lui-même doit évidemment aussi être présent sur la partition racine !

Autre erreur : le séparateur de champs étant le caractère deux points (**:**), tout ce qui traîne en fin de ligne (espace, tabulation, ou autre) sera considéré comme partie intégrante du nom du shell. Ce shell n'existera donc pas, et il sera impossible de se connecter. Ce genre de faute peut être assez difficile à corriger sur certains Unix, où **init** invoque le shell de root et non pas forcément /bin/sh, même en mode mono-utilisateur.

CULTURE **Casser les mots de passe**

Pour casser les mots de passe chiffrés d'Unix, il suffit de tester toutes les combinaisons possibles, de les chiffrer tour à tour, et de comparer les résultats à la chaîne de caractères du fichier /etc/passwd. En y mettant les moyens, on peut désormais trouver en quelques jours un mot de passe chiffré par DES, l'algorithme utilisé historiquement pour les mots de passe d'Unix. On peut restreindre cette méthode de force brute, en rien subtile, à un cas particulier : l'attaque dite « du dictionnaire » se contente de tester les mots et combinaisons simples de mots du dictionnaire dans plusieurs langues.

Les Unix modernes permettent d'utiliser des algorithmes plus résistants, comme MD5, mais le gain de sécurité est discutable. Certes, il faut beaucoup plus de temps pour casser un mot de passe chiffré en MD5 qu'un mot de passe chiffré en DES, mais le succès de l'attaque n'est toujours qu'une question de temps. La meilleure protection reste encore de ne pas dévoiler les mots de passe chiffrés aux usagers autres que root. Les systèmes Unix modernes proposent des dispositifs pour assurer ce niveau de protection.

Sur UNIX System V

Le fichier /etc/passwd doit être lisible par tous, pour permettre par exemple à la commande **`ls -l`** de traduire les UID des propriétaires de fichiers en noms d'utilisateurs. C'est le point faible du système classique, car cela permet aussi à un utilisateur mal intentionné d'avoir accès aux mots de passe chiffrés – il lui est donc possible de tenter une attaque par dictionnaire ; des programmes spécialisés sont même disponibles à cet effet.

Pour remédier à cela, les UNIX System V (et GNU/Linux), proposent le système des *shadow passwords* (mots de passe cachés) : les mots de passe sont stockés dans un autre fichier, /etc/shadow, et les champs de mot de passe du fichier /etc/passwd ne contiennent alors qu'un astérisque.

Le fichier /etc/shadow ne contenant pas d'information utile à tous, on peut en restreindre l'accès à root. Ainsi, /etc/passwd reste accessible à tous, mais les mots de passe sont mieux gardés.

Voici un exemple de fichier /etc/shadow. Le format est similaire à celui du fichier /etc/passwd. Le *login* et le mot de passe chiffré sont suivis de limites de validité du mot de passe (voir la page **`man`** de shadow pour les détails).

```
root:2hKTuNumO4sU6:11565:0:99999:7:::134550460
daemon:*:10946:0:99999:7:::
operator:*:10946:0:99999:7:::
bin:*:10946:0:99999:7:::
news:*:10946:0:99999:7:::
uucp:*:10946:0:99999:7:::
manu:pCs71RItmHgdc:11395:0:99999:7:::134538036
```

ALTERNATIVE **Autres outils pour la gestion des utilisateurs**
On trouve encore des outils de gestion des utilisateurs plus conviviaux, comme **`sysinstall`** ou **`pw`** sur FreeBSD, ou **`sushi`** sur NetBSD, mais ils sont spécifiques à leurs systèmes respectifs.

Sous UNIX System V, on peut toujours modifier /etc/passwd et /etc/shadow avec un éditeur de texte tel que **`vi`**. On peut aussi utiliser des outils qui permettent d'ajouter ou de supprimer des utilisateurs en ligne de commande. Les pages **`man`** des commandes **`useradd`**, **`userdel`** et **`usermod`** fourniront tous les détails. L'édition directe des fichiers évite d'avoir à se souvenir de la syntaxe de ces commandes, mais laisse le champ libre à toute fausse manœuvre. Une méthode intermédiaire utilisant la commande **`vipw`** sera traitée dans la section suivante.

Sur systèmes BSD

La méthode de protection des mots de passe des UNIX System V a un inconvénient : deux fichiers sont nécessaires à la définition des comptes. Lors d'une intervention manuelle, il faut les traiter tous deux, et il est possible d'aboutir à une situation incohérente ou contradictoire (si par exemple **`useradd`** a planté en cours de route).

Les BSD proposent donc une autre approche : un fichier /etc/master.passwd, lisible uniquement par root, qui contient toute l'information des comptes, mots de passe compris. Le fichier /etc/passwd est généré automatiquement à partir de ce dernier et il est lisible par tous. Bien entendu, les mots de passe chiffrés y sont remplacés par des astérisques. Voici un exemple de fichier /etc/master.passwd.

```
root:2hKTuNumO4sU6:0:0::0:0:Charlie Root:/root:/bin/sh
daemon:*:1:31::0:0:The devil himself:/:/sbin/nologin
operator:*:2:5::0:0:System operator:/usr/guest/operator:/sbin/nologin
bin:*:3:7::0:0:Binaries Commands and Source:/:/sbin/nologin
news:*:6:8::0:0:Network News:/var/spool/news:/sbin/nologin
uucp:*:66:1::0:0:Unix-to-Unix Copy:/var/spool/uucppublic:/usr/libexec/uucp/uucico
manu:pCs71RItmHgdc:500:500::0:0:Emmanuel Dreyfus:/home/manu:/bin/ksh
```

On trouve trois champs supplémentaires par rapport à un fichier /etc/passwd traditionnel. Placés entre le GID et le nom complet de l'utilisateur, ils servent à consigner des informations sur la durée de validité du compte ou la classe d'utilisateur telle que définie dans login.conf. Tous les détails sont expliqués dans les pages **man** de master.passwd et login.conf. Ces champs supplémentaires sont absents du fichier /etc/passwd généré à partir de /etc/master.passwd.

Cette approche n'utilise qu'un seul fichier, mais impose l'exécution d'une commande pour reconstruire /etc/passwd à chaque modification de /etc/master.passwd : **pwd_mkdb -p /etc/master.passwd**. C'est peu convivial, mais heureusement, on trouve plus pratique : la commande **vipw**.

vipw invoquera votre éditeur favori, tel que précisé dans la variable d'environnement EDITOR, ou **vi** par défaut, en lui faisant éditer une copie de /etc/master.passwd. À la fin de l'intervention, **vipw** contrôle le format du fichier. S'il est incorrect, il indique l'erreur et propose de ré-éditer le fichier. Si tout va bien, **vipw** remplace /etc/master.passwd par la copie modifiée, puis exécute **pwd_mkdb**. Il s'occupe en outre de verrouiller l'accès à la base de mots de passe pendant l'édition, pour éviter tout accès concurrent.

vipw permet donc d'éditer le fichier de mots de passe en toute sécurité, et il évite d'avoir à se souvenir des détails sordides de la synchronisation des bases de mots de passe sur systèmes BSD. Sur les UNIX System V, dépourvus de fichiers à régénérer, on trouve souvent un programme **vipw** qui permet d'éditer le fichier des mots de passe ; il se contente alors du verrouillage et du contrôle de la syntaxe.

Pour les inconditionnels de la méthode System V, les systèmes BSD proposent des outils de gestion des utilisateurs : **pw** ou **adduser** sous FreeBSD, **user** ou **adduser** sous NetBSD et OpenBSD. Malheureusement, leur syntaxe n'étant pas standardisée d'un système à l'autre, elle est difficile à retenir. On gagne souvent beaucoup de temps à utiliser **vipw**, semblable partout.

PLUS LOIN **Les classes d'utilisateur**

Elles permettent de configurer finement les limites imposées aux utilisateurs. On indique un nom de classe dans le cinquième champ du fichier /etc/master.passwd, et on définit ses attributs dans /etc/login.conf. Il est ainsi possible de définir le nombre maximum de processus autorisés pour un utilisateur, la quantité maximale de mémoire qu'il peut allouer, les priorités de ses processus, des variables d'environnement, etc.
Attention, si le fichier /etc/login.conf existe, tous les utilisateurs doivent appartenir à une classe définie dans ce fichier, sous peine de ne plus pouvoir se connecter au système. Si vous laissez un champ de classe vide dans le /etc/master.passwd, l'utilisateur relèvera de la classe default – qu'il faut définir dans /etc/login.conf.

SUR LES AUTRES UNIX **Commandes BSD sur UNIX System V**

Sur certains UNIX System V, la commande **vipw** se trouve dans le répertoire /usr/ucb, accompagnée d'un certain nombre d'autres utilitaires BSD (notamment un **ps** mode BSD, pour les allergiques au **ps** mode System V).
Le terme ucb signifie « Université de Californie à Berkeley », à l'origine des systèmes BSD.

PLUS LOIN **Les bases binaires**

En réalité, sur les systèmes BSD, les fichiers /etc/master.passwd et /etc/passwd ne sont pas utilisés par les processus ayant besoin de lire des informations relatives aux comptes des utilisateurs. La commande **pwd_mkdb** utilise aussi /etc/master.passwd pour générer deux bases de données binaires des utilisateurs : /etc/spwd.db et /etc/pwd.db. La première, qui contient les mots de passe, est réservée à root ; la deuxième ne contient pas de mot de passe et elle est accessible à tous.

Ces bases binaires permettent un accès plus rapide à l'information que de simples fichiers texte, mais il faut bien noter qu'elles sont uniquement générées à partir du fichier /etc/master.passwd. Si jamais elles devaient être corrompues, il est facile de les régénérer.
Ces bases binaires ont un format qui change suivant que l'on est sur un système petit boutiste ou gros boutiste. Si vous devez migrer une base d'utilisateurs entre deux architectures différentes, n'oubliez pas d'invoquer **vipw** pour régénérer les bases binaires.

Check-list de la création de compte

Créer une entrée dans la base des utilisateurs ne suffit pas, d'autres étapes doivent suivre. Voici une liste non exhaustive des opérations à effectuer lors de la création d'un compte.

- Créer le répertoire personnel de l'utilisateur, y placer les fichiers de configuration par défaut tels que .profile ou .cshrc, et ne pas oublier la commande **chown -R**, qui lui donnera tous les droits sur ses propres fichiers.
- Mettre en place son quota, le cas échéant. C'est la fonction de la commande **edquota**.
- Placer les alias de messagerie dans le fichier /etc/mail/aliases (qui peut se trouver ailleurs selon le système, par exemple sous /etc/aliases), et exécuter la commande **newaliases** pour que la base binaire des alias de messagerie soit mise à jour.

Les programmes de création de compte à la System V ont au moins le mérite d'automatiser tout cela. Mais rien n'empêche d'utiliser conjointement ces programmes et **vipw** pour tirer parti du meilleur des deux mondes.

RAPPEL **chown**

chown permet de changer le propriétaire d'un fichier ou d'un répertoire.

Comment configurer les droits ?

Gérer les groupes

Sous Unix, les groupes permettent de contrôler l'accès à certaines ressources. Chaque utilisateur, on l'a vu, appartient à un groupe principal. Cela suppose donc qu'il peut appartenir à plusieurs groupes...

Le fichier /etc/group établit la correspondance entre les identifiants de groupes, qu'on trouve dans /etc/passwd, et les noms symboliques des groupes, plus mnémotechniques. On y définit aussi les groupes secondaires des utilisateurs. Exemple de fichier /etc/group :

```
wheel:*:0:root,manu
daemon:*:1:daemon
kmem:*:2:root
sys:*:3:root
tty:*:4:root
operator:*:5:root,manu,dumpy
mail:*:6:
bin:*:7:
news:*:8:
guest:*:31:root
nobody:*:39:
users:*:500:
```

PLUS LOIN **Mot de passe de groupe**

Le deuxième champ du fichier /etc/group est aujourd'hui obsolète : il permettait de mettre en place un mot de passe de groupe pour qu'un utilisateur puisse, par le biais de la commande **newgrp**, s'insérer dans un groupe dont il ne faisait pas partie.

SUR LES AUTRES SYSTÈMES **Les ACL**

Traditionnellement, les droits sur les fichiers sous Unix se limitent au propriétaire, à son groupe, et au reste des utilisateurs.
Certains systèmes connaissent des listes de contrôle d'accès (*Access Control Lists*, ou ACL), d'une bien plus grande finesse et expressivité.
Quelques systèmes d'exploitation proposent les ACL, dont Windows NT/2000/XP, Solaris, AIX, TrustedIRIX, et de façon plus expérimentale encore, GNU/Linux et FreeBSD.
Dans le cas des systèmes Unix, les ACL sont standardisées par la norme POSIX 1e.

SUR LES AUTRES UNIX **Netinfo**

Sur NeXTStep et son descendant MacOS X, les fichiers /etc/passwd et /etc/group ne servent qu'au tout début du démarrage, avant le lancement des services **lookupd** et **netinfod**.
Après cette opération, les programmes ayant besoin de renseignements sur les utilisateurs font appel au service **lookupd**, capable d'interroger diverses sources pour trouver les informations demandées. Dans la configuration par défaut, les données concernant les groupes et les utilisateurs sont stockées dans la base Netinfo.
Cette dernière permet de partager facilement une foule d'informations entre plusieurs machines, mais elle est complètement spécifique à MacOS X. C'est le service **netinfod** qui assure l'accès à la base Netinfo. Il répond le plus souvent aux requêtes de **lookupd**, mais on peut également l'interroger à la ligne de commande en tapant **nidump**, pour récupérer par exemple les groupes définis dans la base Netinfo :

```
$ nidump group .
nobody:*:-2:
nogroup:*:-1:
wheel:*:0:root
daemon:*:1:root
kmem:*:2:root
sys:*:3:root
(...)
```

Le format est encore celui de /etc/passwd. Le premier champ contient le nom du groupe, le troisième champ son GID, et le quatrième la liste des utilisateurs qui appartiennent à ce groupe en plus de leur groupe principal.

Dans l'exemple ci-dessus, il est indiqué entre autres que le groupe de GID 500 a pour nom `users`. Dans le fichier /etc/passwd donné plus haut, l'utilisateur `manu` avait pour GID 500 : il appartient donc au groupe `users`, et il n'est pas utile de préciser cela à nouveau dans le fichier /etc/group. En revanche, cet extrait précise que `manu` a pour groupes secondaires `wheel` et `operator`.

Le fichier /etc/group n'est pas utilisé pour produire de base binaire, et même endommagé, il n'empêchera pas root de se connecter sur la console de la machine. On peut donc le manipuler sans craintes avec un éditeur de texte. Il existe toutefois des commandes spécialisées pour le modifier, mais ici encore leur syntaxe varie beaucoup d'un système à l'autre.

Attributs spéciaux *set-UID* et *set-GID*

Les attributs spéciaux *set-UID* et *set-GID*, qui concernent les exécutables, jouent un grand rôle dans la gestion des droits sur un système Unix : ils modifient les droits avec lesquels s'exécute le programme.

En temps normal, lorsqu'un utilisateur invoque un programme, ce dernier est exécuté avec les droits de cet utilisateur : il porte donc son UID et son GID. Mais un programme *set-UID* et/ou *set-GID* prendra respectivement lors de son exécution l'UID du propriétaire du fichier contenant le programme et/ou le GID de son groupe.

Ce mécanisme permet par exemple aux utilisateurs de modifier leurs mots de passe avec la commande **passwd**. Cette opération requiert en effet des droits en écriture sur le fichier /etc/master.passwd, droits dont seul root dispose. Mais l'exécutable **passwd**, qui appartient à root, est *set-UID* (on parle de programme *set-UID* root) ; il s'exécute donc avec les privilèges de root, ce qui lui permet de remplir sa fonction.

SÉCURITÉ **Attention aux binaires set-UID**

Les programmes *set-UID* sont pratiques, mais dangereux. Un exécutable *set-UID* root doit être très soigneusement programmé, pour éviter toute exploitation abusive par un utilisateur peu scrupuleux (c'est une source classique de failles de sécurité).
Placer un attribut *set-UID* sur un programme quelconque peut être lourd de conséquences. Un /bin/sh *set-UID* donnera par exemple un shell avec les droits de root, permettant ainsi à un utilisateur de tout faire sur le système. Dans le cas général, ne donnez pas l'attribut *set-UID* ou *set-GID* à un programme qui n'a pas été conçu pour cela.

Ces attributs spéciaux se manipulent avec la commande **chmod**, et sont matérialisés par un `s` en lieu et place du `x` dans l'affichage des droits donné par **`ls -l`**. Quelques exemples seront plus parlants :

```
# ls -l toto
-r-sr-sr-x  1 root  daemon    23248 Apr  1 17:57 toto
# chmod ug-s toto ❶
# ls -l toto
-r-xr-xr-x  1 root  daemon    23248 Apr  1 17:57 toto
# chmod u+s toto ❷
# ls -l toto
-r-sr-xr-x  1 root  daemon    23248 Apr  1 17:57 toto
# chmod u-s toto ❸
# ls -l toto
-r-xr-xr-x  1 root  daemon    23248 Apr  1 17:57 toto
# chmod g+s toto ❹
# ls -l toto
-r-xr-sr-x  1 root  daemon    23248 Apr  1 17:57 toto
```

❶ On supprime les deux attributs

❷ On met en place l'attribut *set-UID*

❸ On ôte l'attribut *set-UID*

❹ On met en place l'attribut *set-GID*

PLUS LOIN **Scripts set-UID root**

Le système n'honore pas les attributs *set-UID* et *set-GID* sur les scripts. L'interpréteur correspondant (souvent un shell dans le cas de scripts shell) sera donc invoqué avec les droits habituels.
Cette limitation est volontaire, car les scripts *set-UID* sont dangereux. Un script héritant des variables d'environnement de l'utilisateur, il existe de nombreux moyens de détourner son comportement et compromettre ainsi la sécurité du système. La manière la plus simple est de fournir au script une variable `PATH` lui faisant exécuter des scripts personnels au lieu de commandes classiques. Prenons le cas du script suivant, nommé killxlock.sh, et qui a pour effet de tuer un éventuel processus de verrouillage d'écran :

```
#!/bin/sh
ps -ax | grep xlock | awk '{print $1}' | xargs kill
```

Il est aisé de détourner son comportement : en plaçant /tmp en tête du `PATH` et en y créant le fichier /tmp/ps suivant :

```
#!/bin/sh
chmod uog+r /etc/shadow
```

L'exécution de killxlock.sh avec les droits de root rendra alors /etc/shadow lisible par tous. Cet exemple est facilement transposable à tout script shell, et c'est pourquoi ces attributs spéciaux ne sont pas honorés sur les scripts.
Il est possible d'exécuter un script shell avec des droits non classiques, mais il faut pour cela l'envelopper dans un programme écrit en `C`, tel que celui-ci :

```
/* Compilation: cc -o wrapper wrapper.c */

#include <unistd.h>
#include <errno.h>
#include <err.h>

int
main(ac, av)
        int ac;
        char **av;
{
        char *argv[] = { "/usr/local/bin/killxlock.sh", NULL };
        char *envp[] = { NULL };
        int error;

        setuid(geteuid());
        if ((error = execve(argv[0], argv, envp)) != 0)
                err(errno, "execve failed");
        return 0;
}
```

Le système acceptera d'exécuter ce programme enveloppant (*wrapper* en anglais) avec les privilèges adéquats s'il est *set-UID*. Ce petit code se contente d'invoquer le script grâce à l'appel système **`execve()`**, mais sans aucune variable d'environnement. Rapportez-vous à la page **man** de la fonction **execve** pour obtenir plus de détails.
Le script exécuté en bout de chaîne ne recevant aucun environnement, il faudra systématiquement y définir des variables comme `PATH` ou `HOME`. De plus, on le rédigera avec le plus grand soin s'il doit interagir avec l'utilisateur, pour éviter tout détournement de son comportement.

PLUS LOIN **Droit `S` au lieu de `s`**

Vous verrez parfois des fichiers affichant le droit `S` au lieu de `s`, comme ceci :

```
$ ls -l lpr
-r-sr-Sr-x  1 root  daemon    23248 Apr  1 17:57 lpr
```

Cette situation correspond au cas où l'attribut *set-UID* ou *set-GID* est positionné alors que le droit en exécution ne l'est pas. L'attribut *set-UID* ou *set-GID* n'a alors pas de sens.

Groupes spéciaux pour utilisateurs spéciaux

En général, les privilèges particuliers d'un groupe ou d'un utilisateur sont liés à ses droits dans le système de fichiers. Par exemple, les membres du groupe operator peuvent effectuer les sauvegardes car ils peuvent lire les fichiers spéciaux donnant accès aux disques :

```
$ ls -l /dev/sd0a
brw-r-----  1 root  operator  4, 0 Jul 28  2001 /dev/sd0a
```

Seul l'utilisateur d'UID zéro échappe à cette règle : il a les privilèges d'administrateur. Il s'appelle souvent, par convention, root, mais ce n'est ni obligatoire ni lié à ses super-pouvoirs : seul l'UID compte. Certains s'amusent même à lui donner un autre nom, en jouant sur les mots ou les allusions (**`route`**, ou **`kgb`**).

Tous les autres pouvoirs particuliers accordés aux utilisateurs sont liés aux permissions dans le système de fichiers : droits des fichiers spéciaux du /dev, ou attributs *set-UID* et *set-GID* positionnés sur des commandes auxquelles l'utilisateur a accès.

Accession au pouvoir suprême

La commande **`su`**, qui est *set-UID* root, permet d'obtenir un shell avec les droits de root – à condition bien sûr d'en fournir le mot de passe, qui sera dûment contrôlé.

Sur les systèmes BSD, cette commande vérifie plus de choses que sur les autres Unix : **`su root`** est réservée aux membres du groupe `wheel`, de GID zéro. **`su`** rejettera immédiatement, sans même leur demander de mot de passe, tous les autres utilisateurs, avec le message :

```
$ su root
su: you are not listed in the correct secondary group (wheel) to su root.
```

Traduction : vous n'appartenez pas au groupe secondaire adéquat (`wheel`) pour passer root via **`su`**.

Cette contrainte est extrêmement confortable : elle assure que même après découverte du mot de passe de root, un utilisateur indélicat ne pourrait pas l'exploiter par la commande **`su`**. Cela ne garantit pas une tranquillité totale : les connexions de root à distance ou sur la console sont encore possibles, à moins que la machine n'ait été configurée pour les refuser.

CULTURE **L'utilisateur toor**

Certains systèmes Unix ont un deuxième utilisateur d'UID zéro, qui s'appelle toor. Son compte est par défaut désactivé, mais on peut lui donner un mot de passe si on souhaite l'utiliser. Cela permet d'avoir un compte administrateur de secours en cas d'incident empêchant root de se connecter : shell effacé par erreur, mot de passe oublié…

Pour être vraiment efficace, le compte toor doit avoir un shell et un répertoire personnel différents de ceux de root (on utilise généralement /bin/sh et /altroot).

PIÈGE À C… **Description du groupe wheel**

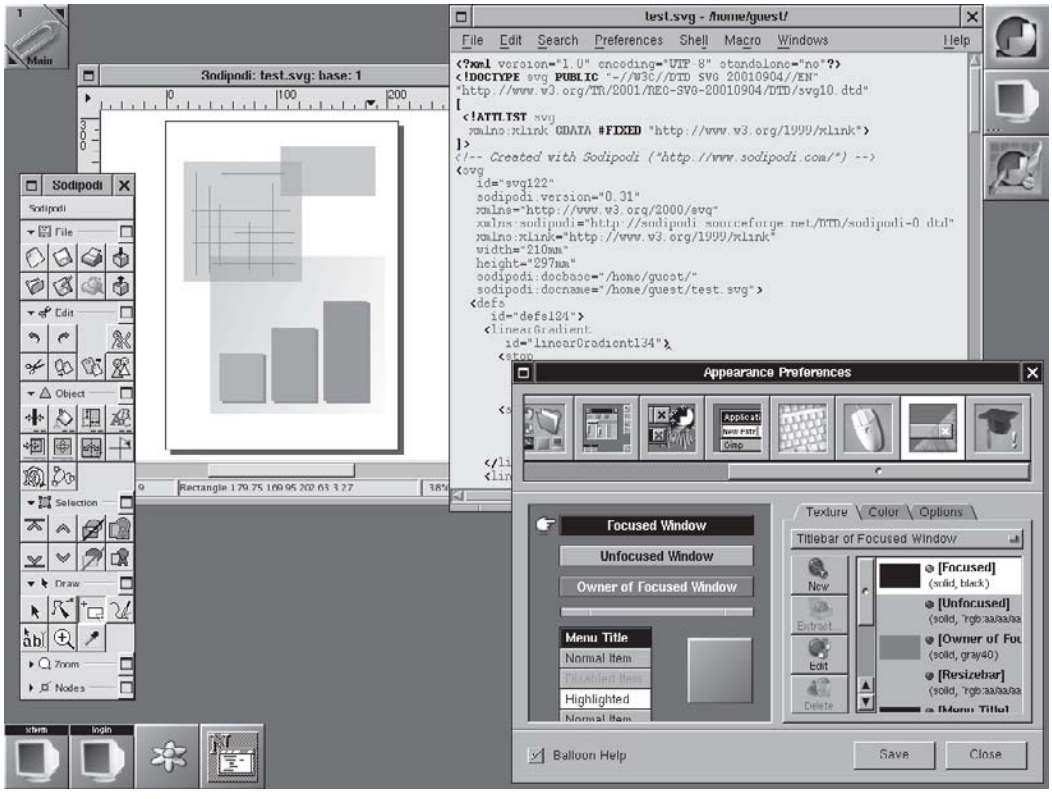
On peut forcer le **`su`** des systèmes BSD à se comporter comme le **`su`** traditionnel : si le groupe `wheel` est vide ou inexistant dans /etc/group, n'importe qui pourra taper **`su root`**. C'est pour éviter cet écueil que par défaut, le fichier /etc/group d'un BSD explicite que root appartient au groupe `wheel`, même si cela est déjà indiqué dans le fichier /etc/passwd.

Celui qui pense bien faire en délestant le fichier /etc/group de cette ligne à ses yeux inutile induira donc un effet de bord inattendu et peu souhaitable.

ASTUCE **`su` et les fichiers de configuration du shell**

Si vous avez soigneusement mis en place une configuration dans les fichiers .profile ou .cshrc de root, vous serez probablement déçu en constatant que le shell invoqué par **`su`** les ignore. Le shell issu de **`su`** lira ses fichiers de configuration si l'on a fourni l'option **`-l`** en tapant **`su -l`** (*login shell*, ou shell de connexion).

7


test.svg - /home/guest/
File Edit Search Preferences Shell Macro Windows Help
Sodipodi: test.svg: base: 1
Sodipodi
Appearance Preferences
Focused Window
Unfocused Window
Owner of Focused Window
Menu Title
Normal Item
Highlighted
Texture Color Options
Titlebar of Focused Window
[Focused]
(solid, black)
[Unfocused]
[Resizebar]
Balloon Help
Save
Close

Configuration d'une station Unix

La configuration d'une station Unix passe par certaines étapes classiques que l'on retrouve très souvent : le réseau, l'interface graphique, et les imprimantes.

SOMMAIRE

MOTS-CLEFS

- Réseau
- Interface graphique
- Imprimantes

Certaines des opérations décrites dans le présent chapitre s'appliquent assez facilement à tout système Unix, d'autres sont spécifiques aux BSD. Comme dans les chapitres précédents, nous présenterons les choses d'une façon la plus générique possible. Malheureusement, il n'y a pas toujours de standard, notamment quand chaque système adopte ses propres conventions.

Configuration du réseau

Traiter exhaustivement de la configuration du réseau est une tâche difficile. Les situations sont multiples et la situation empire avec l'arrivée de nouveaux types de réseaux, que chaque système gère à sa manière.

Nous aborderons rapidement les situations les plus usuelles : connexion à un réseau Ethernet, avec ou sans configuration automatique par DHCP, connexions à l'ADSL en PPPoE, et connexion par modem RTC.

Connexion à un réseau Ethernet, configuration manuelle

C'est la situation la plus courante, celle du raccordement sur un réseau d'entreprise, d'université, ou domestique. Il faut connaître les informations suivantes, qui pourront être fournies par l'administrateur réseau :

- Adresse IP de la machine
- Masque de sous-réseau
- Adresse IP de la passerelle par défaut
- Adresses IP du ou des DNS

Où est passée ma carte réseau ?

Muni de ces indispensables informations de configuration, il faut configurer la carte Ethernet. C'est la commande **`ifconfig`** qui permet de configurer les interfaces réseau d'une machine Unix, qu'elle identifie chacune par un nom.

B.A.-BA **Le DNS**

Le DNS est l'annuaire d'Internet, il permet de traduire des adresses symboliques DNS telles que **`www.apple.com`** en adresses IP telles que `17.254.0.91`, et inversement (on parle alors de *reverse DNS*, ou DNS inverse).

EN PRATIQUE **Et si c'est vous l'administrateur réseau ?**

Si vous êtes l'administrateur réseau, vous savez déjà quelles adresses vous attribuer. Si vous hésitez sur les adresses à utiliser dans le cadre d'un réseau domestique, utilisez les paramètres suivants :

- Adresses IP : `192.168.0.1` et suivantes (`192.168.0.2`, etc.).
- Masque de sous-réseau : `255.255.255.0`.
- Passerelle par défaut : rien pour le moment.
- DNS : également en attente

La passerelle par défaut et le DNS seront positionnés lors du raccordement à Internet.

B.A.-BA **Trop d'acronymes**

Les acronymes abondent déjà. Quelques définitions seront sans doute les bienvenues.

- DHCP (*Dynamic Host Configuration Protocol*, ou protocole de configuration automatique de l'hôte), permet de configurer automatiquement les machines sur le réseau.
- ADSL (*Asymmetric Digital Subscriber Line*, ou ligne d'abonnement numérique asymétrique – l'équivalent français officiel est RNA pour Raccordement Numérique Asymétrique) est un service de raccordement à Internet exploitant la bande passante inutilisée des lignes téléphoniques. Il peut fonctionner avec plusieurs protocoles, dont PPPoE (*Point to Point Protocol over Ethernet*, ou protocole point à point sur Ethernet).
- RTC (Réseau Téléphonique Commuté) désigne le réseau téléphonique analogique classique, utilisé pour téléphoner et parfois pour se connecter à Internet.

Pour configurer la carte Ethernet, il faut donc connaître le nom de son interface. **`ifconfig -a`** affiche toutes les interfaces disponibles :

```
$ ifconfig -a
ne2: flags=8863<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        address: 00:00:e8:ea:d1:e2
        media: Ethernet autoselect (10baseT)
        inet 10.0.4.23 netmask 0xffffff00 broadcast 10.0.4.255
        inet6 fe80::200:e8ff:feea:d1e2%ne2 prefixlen 64 scopeid 0x1
lo0: flags=8009<UP,LOOPBACK,MULTICAST> mtu 33220
        inet 127.0.0.1 netmask 0xff000000
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x2
ppp0: flags=8010<POINTOPOINT,MULTICAST> mtu 1500
ppp1: flags=8010<POINTOPOINT,MULTICAST> mtu 1500
sl0: flags=c010<POINTOPOINT,LINK2,MULTICAST> mtu 296
sl1: flags=c010<POINTOPOINT,LINK2,MULTICAST> mtu 296
strip0: flags=0 mtu 1100
strip1: flags=0 mtu 1100
```

Cet exemple dispose des interfaces suivantes : `ne2`, `lo0`, `ppp0`, `ppp1`, `sl0`, `sl1`, `strip0`, et `strip1`. `ne2` est une carte Ethernet, comme précisé dans la ligne `media`. Les autres éléments sont des pseudo-interfaces ; nous nous y intéresserons plus tard.

Y a-t-il un pilote pour sauver ma carte ?

Quand on ne trouve pas l'interface correspondant à une carte, c'est souvent que le noyau ne contient pas de pilote pour cette dernière. Ce problème se diagnostique au démarrage par un message du type :

```
mc at obio0 offset 0x11000 not configured
```

Le pilote peut encore exister mais ne pas être inclus dans le noyau générique, installé par défaut ; il faudra dans ce cas recompiler un noyau en l'y activant. Il arrive aussi que la carte soit trop récente pour que le pilote existe dans la version de production du système. On peut alors faire appel à la version de développement du noyau (*-current*) pour en bénéficier.

Dernière possibilité : il n'existe pas de pilote pour la carte envisagée. On peut alors mettre une casquette de développeur et tenter de l'écrire, ou prêter la carte à un développeur qui en est capable. N'oubliez pas que pour écrire un pilote il faut disposer du matériel, et les projets BSD, bénévoles, n'ont pas la trésorerie nécessaire pour acheter toutes les nouvelles cartes. Entre-temps, il faudra trouver une carte reconnue par le système pour pouvoir utiliser le réseau.

PLUS LOIN **Passage en -current**

-current est la version de développement du système, appelée à en devenir la prochaine version stable. Pour NetBSD, on peut choisir d'y passer tout le système ou son seul noyau, en conservant le reste du système en l'état. Sous FreeBSD et OpenBSD, la mise à jour du noyau doit obligatoirement être accompagnée de celle du reste du système sous peine de grave dysfonctionnements.
La recompilation du noyau sera abordée en détails au chapitre 13.

VOCABULAIRE **Pseudo-interfaces**

Alors que les interfaces réseau « normales » correspondent à une carte réseau, les pseudo-interfaces ne sont liées qu'à une couche logicielle.
Par exemple, les interfaces `ppp` sont mises en place par le protocole PPP. qui sert le plus souvent à exploiter une connexion via un modem RTC. Dans ce cas, PPP utilise un port série, lié à l'interface `ppp` au moment de la configuration. Le port série lui-même n'étant pas a priori une interface réseau, il n'apparaît pas dans la sortie de **`ifconfig -a`**.

RAPPEL **Messages de diagnostic**

Les messages de diagnostic du démarrage défilent vite, mais on peut les relire à tout moment sans devoir redémarrer la machine, grâce à la commande **`dmesg | less`**.

ATTENTION **ping ne fonctionne pas forcément tout de suite**

Les premiers **ping** ne passent pas forcément tout de suite : on observe parfois un délai de latence de quelques secondes. Ceci est dû aux requêtes ARP, dont nous parlerons au chapitre 9.

ASTUCE **Option -n**

L'option **-n** de **ping** désactive la résolution de noms, dont elle évite d'attendre l'échec de vaines tentatives lorsque le DNS n'est pas correctement configuré.

RAPPEL **Control+C**

^C correspond à une combinaison **Control+C** enfoncée par l'utilisateur, en général utilisée pour interrompre une commande (en lui envoyant le signal `SIGTERM`).

La configuration proprement dite

Quand on connaît le nom de l'interface de la carte, on peut passer à l'acte et configurer le réseau. Voici un exemple de configuration d'une interface nommée `ex0` :

```
# ifconfig ex0 inet 192.168.3.21 netmask 255.255.255.0
```

Elle reçoit l'adresse `192.168.3.21` et le masque de sous-réseau `255.255.255.0`. En présence d'une autre machine sur le même réseau local, on peut immédiatement tester que tout fonctionne avec la commande **ping** :

```
# ping -n 192.168.3.1
PING 192.168.3.1 (192.168.3.1): 56 data bytes
64 bytes from 192.168.3.1: icmp_seq=0 ttl=255 time=5.097 ms
64 bytes from 192.168.3.1: icmp_seq=1 ttl=255 time=2.889 ms
64 bytes from 192.168.3.1: icmp_seq=2 ttl=255 time=2.863 ms
^C
----10.0.12.1 PING Statistics----
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 2.863/3.616/5.097/1.282 ms
```

Passerelle par défaut et DNS

La commande **route** permet de configurer la passerelle par défaut :

```
# route add default 192.168.3.1
add net default: gateway 192.168.3.1
```

On la teste en exécutant **ping** et en ciblant une machine d'un autre sous-réseau (sans oublier l'option **-n** si le DNS n'est pas encore correctement configuré).

EN CAS DE COUP DUR **ping ne fonctionne pas**

Si **ping** échoue, le problème peut venir de la machine source comme de la machine qu'il cible. Assurez-vous que cette dernière fonctionne bien, que les adresses IP des machines sont sur la même plage (c'est-à-dire qu'elles commencent par les mêmes numéros), et que les masques de sous-réseau sont identiques. Désactivez d'éventuels dispositifs de filtrage de la machine cible pouvant bloquer le **ping**.
Si tout est correct et que **ping** s'entête à échouer, il faut diagnostiquer la panne. Plusieurs outils le permettent :
Si le *hub* (concentrateur, ou répéteur) a des témoins d'activité, ils permettent normalement d'observer chaque **ping** et sa réponse. Si rien ne clignote, la carte n'émet rien. Si les diodes trahissent malgré tout le passage du **ping** et de sa réponse, c'est que la carte ne reçoit rien.
Autre outil de diagnostic, la commande **tcpdump** permet de voir passer les paquets sur une interface donnée.

```
# tcpdump -i ex0
tcpdump: listening on ex0
13:31:33.234631 192.168.3.21 > 192.168.3.1: icmp: echo request seq 0
13:31:33.234695 192.168.3.1 > 192.168.3.21: icmp: echo reply seq 0
(...)
```

Si le système n'est pas capable d'émettre ou de recevoir des paquets, contrôlez-en les connexions physiques (connecteurs, câbles). Si tout est correct, il se peut que la carte ne soit pas correctement prise en charge par le noyau, ce qui peut avoir différentes explications :

- Problème de conflit d'IRQ ou de port d'entrée/sortie (spécifique aux PC). La commande **dmesg | less** devrait permettre de découvrir avec quel matériel la carte est en conflit.
- Détection incorrecte de la carte. Les fonctionnalités *Plug'N Play*, censées assister, introduisent parfois des problèmes. Tentez d'activer ou de désactiver le *Plug'N Play* (spécifique aux PC encore).
- Bogue dans le pilote de la carte. Tentez dans ce cas de passer le noyau (ou tout le système) en *-current*.
- Si le problème persiste en *-current*, changez de carte (et faites un rapport de bogue).

Le fichier /etc/resolv.conf configure le DNS. Exemple :

```
nameserver 192.168.3.1
```

Pour en tester le bon fonctionnement, il suffit d'invoquer **ping** sur une machine par nom DNS et non pas par IP.

Configuration au démarrage

Comment configurer automatiquement le réseau à chaque démarrage ? Les moyens proposés diffèrent hélas d'un système à l'autre, même entre systèmes BSD. Pour ces derniers, tout est détaillé dans la page **man** de rc.conf.

Sous NetBSD et FreeBSD, on peut par exemple inscrire dans /etc/rc.conf la ligne suivante pour configurer automatiquement l'interface `ex0` :

```
ifconfig_ex0="inet 192.168.3.21 netmask 255.255.255.0"
```

La route par défaut peut elle aussi se configurer via ce fichier. On la précise dans la variable `defaultrouter` sous FreeBSD, `defaultroute` sous NetBSD. Exemple d'utilisation :

```
defaultroute="192.168.3.1"
```

La configuration du DNS étant toujours lue dans le fichier /etc/resolv.conf, elle sera préservée au redémarrage. Elle a de plus le bon goût d'avoir la même syntaxe sur pratiquement tous les Unix.

Auto-configuration du réseau avec DHCP

Si le réseau dispose d'un serveur DHCP (comme dans le cas d'une connexion câble), la situation est nettement plus confortable : il suffit d'invoquer le client DHCP, qui configurera tout automatiquement.

Il s'appelle **dhclient**, et provient sur presque tous les systèmes Unix de l'*Internet Software Consortium* (ISC) ; son comportement est donc assez semblable d'un système à l'autre.

On peut lui indiquer une interface à configurer ou l'exécuter sans argument, auquel cas il tentera de configurer toutes les interfaces disponibles.

PIÈGE À C... **Tester la configuration**

En cas de doute lors de toute modification de la configuration du réseau au démarrage, il faut redémarrer la machine.
Les machines Unix n'ont pas besoin de redémarrer souvent ; il peut donc s'écouler plusieurs semaines entre une intervention sur les fichiers de configuration et le prochain démarrage. Si la configuration est erronée, vous risquez de devoir la corriger longtemps après l'avoir faite, ce qui n'est pas toujours facile, car entre-temps vous aurez oublié ce que vous aviez modifié.

ASTUCE **Nettoyer les routes**

La commande **route flush** se débarrasse d'éventuelles routes saisies par erreur.

SUR LES AUTRES UNIX **Syntaxe de la commande route**

Elle varie un peu d'un Unix à l'autre. Sous GNU/Linux, le mot-clef **gw** est obligatoire avant d'indiquer la passerelle. On écrira ainsi **route add default gw 192.168.0.1**.

SUR LES AUTRES UNIX **Fichiers de configuration du réseau**

L'emplacement des fichiers de configuration varie beaucoup d'un système à l'autre. Face à un Unix inconnu dont le réseau est déjà configuré, on trouvera l'emplacement de la configuration réseau avec un **grep** récursif de l'adresse IP, du masque de sous-réseau, et de la passerelle par défaut, dans le répertoire /etc. Exemple : **grep -r 192.168.0.1 /etc**.

ALTERNATIVE **Configuration du réseau**

D'autres fichiers que /etc/rc.conf permettent de configurer le réseau. Sous NetBSD et OpenBSD, on peut par exemple définir la passerelle par défaut dans /etc/mygate et les configurations des interfaces dans des fichiers /etc/ifconfig_xxx, où xxx représente le nom de l'interface.

```
# cat /etc/mygate
192.168.3.1
# cat /etc/ifconfig_ex0
inet 192.168.3.21 netmask 255.255.255.0
```

Pour fonctionner correctement, **dhclient** exigera de pouvoir écrire dans /etc/resolv.conf, fichier contenant la configuration du DNS. Cela posera des problèmes si la racine est montée en lecture seule. La façon la plus simple résoudre cela est de monter cette dernière en lecture/écriture. Si c'est impossible (démarrage sur CD-ROM, système de fichiers endommagé, ...), on peut avoir recours à une astuce (syntaxe valable sur NetBSD et OpenBSD) :

```
# mount -t mfs swap /tmp
# mkdir /tmp/etc
# mount -t union /tmp/etc /etc
# touch /tmp/etc/resolv.conf
```

On crée ici un système de fichiers en mémoire, que l'on monte sur /tmp. On profite de cette zone inscriptible pour créer /tmp/etc, que l'on remonte sur /etc, en union. Cette manipulation a pour conséquence de rendre visibles les fichiers du répertoire /tmp/etc par-dessus le répertoire /etc. Il reste à créer un /tmp/etc/resolv.conf, qui apparaîtra au-dessus du /etc/resolv.conf original. Cette méthode est applicable à d'autres systèmes que NetBSD et OpenBSD, mais sûrement pas avec la même syntaxe.

PLUS LOIN **Montage de systèmes de fichiers en mémoire**

Cette recette peut aussi servir après une destruction accidentelle du fichier /etc/fstab sur un système qui l'exige pour monter la racine en lecture/écriture. Un tel montage étant indispensable pour pouvoir créer le /etc/fstab manquant, l'administrateur doit faire face à un problème d'œuf et de poule.
En montant un système de fichiers en mémoire en union sur /etc, il pourra créer le fichier /etc/fstab et résoudre son problème.

Bien entendu, **dhclient** peut être invoqué automatiquement au démarrage, par une ligne **dhclient=YES** dans /etc/rc.conf sur NetBSD, et par un lien dans un répertoire /etc/rc.d sur les UNIX System V et GNU/Linux. Sous FreeBSD, on obtient l'invocation du client DHCP au démarrage en indiquant le mot-clef **DHCP** à la place de l'adresse de l'interface dans /etc/rc.conf :

```
ifconfig_fxp0="DHCP"
```

Connexion par modem RTC

L'utilisation d'un modem pour se connecter à Internet par ligne téléphonique était le cas le plus courant il y a quelques années, mais il cède désormais le haut du pavé aux connexions via le câble ou l'ADSL, proposant de meilleurs débits, et tarifées forfaitairement.

La facturation est en effet, en France, le gros problème des connexions téléphoniques classiques : chaque tentative, même ratée, se paie. La configuration n'étant pas très simple, c'est financièrement un mauvais moment à passer.

Où est connecté le modem ?

Sous Unix, on peut interagir avec un modem via un fichier spécial du répertoire /dev. Seul souci : ce répertoire contient des centaines de fichiers, et celui qu'il faut utiliser dépend du port de branchement du modem.

Si le modem est connecté à un port série, ce fichier sera probablement /dev/tty00 ou /dev/tty01, mais cela peut varier sur certaines plateformes exotiques. Les modems connectés en USB peuvent aussi être accessibles par des noms différents.

ATTENTION **Modems internes**
Les modems internes sont rarement reconnus par les systèmes Unix libres, faute de documentation. En particulier, qui dispose d'un Win-modem sur PC devra sans doute se procurer un modem externe.

Si le modem allume quelques diodes après la commande **cat > /dev/tty00**, c'est qu'il s'agit du bon fichier spécial. Dans le cas contraire, on ne peut rien conclure, car les modems ont des comportements assez divers.

Le *daemon* pppd

Sous BSD, c'est le *daemon* **pppd** qui assure les connexions par modem RTC. Il utilise la commande **chat** pour dialoguer avec le modem et initier la connexion. Il assure ensuite la phase d'authentification, configure une pseudo-interface ppp qu'il relie au fichier spécial où est branché le modem, et il ne lui reste plus qu'à configurer la route par défaut et le DNS indiqués par le fournisseur d'accès Internet.

Le *daemon* **pppd** se configure via des fichiers situés dans /etc/ppp/peers – on peut en créer un par fournisseur d'accès. Exemple de fichier /etc/ppp/peers/isp, extrait de la page **man** de **pppd** (dont la lecture est conseillée) :

```
tty00 ❶
56000 ❷
crtscts ❸
usepeerdns ❹
noauth ❺
connect '/usr/sbin/chat -v -f /etc/ppp/chat-isp' ❻
```

On trouve entre autres dans ce fichier :

❶ Le nom du fichier spécial de /dev qui permet l'accès au modem.

❷ La vitesse de connexion en bits par secondes.

❸ Le type de contrôle de flux. crtscts convient la plupart du temps, sauf cas particulier : sur Macintosh, il faut utiliser cdtrcts quand on utilise un port série intégré.

❹ Indique à **pppd** de modifier /etc/resolv.conf pour y ajouter les DNS fournis lors de la configuration de PPP.

❺ Par défaut, **pppd** demande à la machine distante de s'authentifier, ce qui déplaît en général aux centres d'appel des fournisseurs d'accès. Cette option désactive cette demande.

❻ L'emplacement d'un fichier /etc/ppp/chat-isp utilisé par la commande **chat** pour démarrer la connexion.

SUR LES AUTRES UNIX **Configuration du DNS**

Windows dispose d'extensions au protocole PPP permettant de configurer automatiquement le DNS lors de la mise en place d'une configuration PPP. Le *daemon* **pppd** des systèmes Unix a été modifié pour prendre en compte cette option, absente des Unix les plus anciens. Dans le cas classique, il faut connaître les DNS de son fournisseur d'accès Internet et les placer à la main dans /etc/resolv.conf.

SÉCURITÉ **Option –v pour chat**

L'option **–v** de **chat** provoque un affichage verbeux, très utile pour comprendre les problèmes qui surviendront aux premières connexions. Ces messages apparaissent normalement dans /var/log/messages.
Pour pratiques qu'ils soient, ces messages ont l'inconvénient majeur de contenir le mot de passe, consigné dans un journal lisible par tous. Une fois la connexion au point, pensez donc à supprimer l'option **–v** et à faire le ménage dans /var/log/messages.

PLUS LOIN **Nom des fichiers spéciaux**

Trouver le bon fichier spécial pour un périphérique donné peut se révéler difficile. Ces fichiers sont caractérisés par deux numéros, majeur et mineur, affichés par **ls –l**.

```
$ ls -l /dev/tty0*
crw-------  1 root  wheel  12, 0 May  9 07:27 /dev/tty00
crw-------  1 root  wheel  12, 1 Jul 27  2001 /dev/tty01
```

Le c de la première colonne indique un fichier spécial. Les numéros majeurs sont ici 12 dans les deux cas ; les mineurs sont 0 et 1.

Le majeur indique à quel pilote correspond le fichier. Lors des accès au fichier spécial, le noyau utilise une table mettant en correspondance les numéros majeurs et les pilotes. Quand on ne trouve pas à quel fichier spécial correspond un matériel, en désespoir de cause et en l'absence de documentation, c'est cette table qu'il faut consulter dans les sources du noyau.
Le mineur permet au même pilote de gérer plusieurs périphériques.

PLUS LOIN **Fichiers pour la commande `chat`**

Le modem et l'ordinateur dialoguent avec un jeu de commandes dites commandes Hayes : l'ordinateur envoie une requête et le modem y répond. La commande **`ATDT`** permet par exemple de composer un numéro de téléphone. Si la connexion se fait correctement, le modem répondra `CONNECT`.
Les lignes **`ABORT`** servent à indiquer certaines réponses du modem considérées comme fatales. En recevant de telles réponses, **`chat`** abandonnera la connexion.

PLUS LOIN **CHAP et PAP**

CHAP et PAP sont deux protocoles d'authentification utilisés dans le cadre de PPP. Ils utilisent respectivement les fichiers d'informations de connexion (nom d'utilisateur et mot de passe) /etc/ppp/chap-secret et /etc/ppp/pap-secret. Pour éviter de dupliquer l'information, on peut mettre en place un lien de l'un vers l'autre. Ces fichiers contenant des mots de passe, ils doivent appartenir à root et n'être lisibles que par lui.
Certains serveurs d'authentification n'utilisent ni CHAP ni PAP, mais fonctionnent par invite de connexion. C'est à **`chat`** de traiter le problème ; des exemples sont donnés dans la page **`man`**.

RAPPEL **`ps` et `kill`**

Nous avons vu ces commandes au chapitre 3. Sur un système BSD, **`ps -ax`** permet d'obtenir la liste des processus, et **`kill`** leur envoie des signaux. Il a pour argument le PID du processus cible et sans option particulière, le tuera.
Profitons-en pour placer une petite ligne sympathique, qui tuera automatiquement le processus **`pppd`** quel que soit son PID (identifiant de processus) : **`ps -ax | awk '/pppd/{print $1}' | xargs kill`**

Les scripts `chat`

C'est dans le fichier lu par **`chat`** que l'on place le numéro de téléphone du fournisseur d'accès. Voici un exemple de fichier chat-isp utilisant le numéro 01 53 36 66 66 :

```
ABORT BUSY
ABORT "NO CARRIER"
ABORT "NO DIALTONE"
REPORT CONNECT
'' ATZ
OK ATDT0153366666
CONNECT ''
```

Authentification

Dernier point, il faut indiquer le nom d'utilisateur (*login*) et le mot de passe dans le fichier /etc/ppp/chap-secret ou /etc/ppp/pap-secret selon le protocole d'authentification utilisé. Ces fichiers ont le même format. Exemple pour le *login* **`toto`** avec mot de passe **`q`** :

```
toto    *       q
```

Connexion au serveur d'appel

Une fois tout ceci mis en place, on peut tenter de se connecter en exécutant **`pppd call`** suivi du nom du fichier de configuration du répertoire /etc/ppp/peers :

```
# pppd call isp
```

Il est peu probable que tout fonctionne du premier coup. Les messages d'erreur, consignés dans /var/log/messages, devraient permettre de résoudre les problèmes qui surviendront.

Après établissement de la connexion, la commande **`ifconfig ppp0`** indiquera l'adresse IP attribuée. La déconnexion se fera simplement en tuant le *daemon* **`pppd`** avec la commande **`kill`**.

Connexion ADSL avec PPPoE

Les connexions ADSL sont de plus en plus répandues. Suivant les choix faits par le fournisseur d'accès Internet, elles utilisent des protocoles différents, le plus souvent PPPoE, PPTP, ou PPPoA. PPPoE est de loin le cas le plus courant, c'est donc sur lui que portera notre exemple.

PPPoE assure une connexion authentifiée à travers Ethernet. C'est un protocole récent, et chaque système Unix a sa méthode pour l'exploiter. Même les systèmes BSD ont chacun leur méthode officielle, ainsi que d'autres, officieuses. Il est donc difficile d'exposer une documentation exhaustive sur le sujet. À titre d'exemple, examinons rapidement la configuration d'une connexion ADSL avec PPPoE sous NetBSD et sous FreeBSD.

Configuration de PPPoE sous NetBSD

Dans une certaine mesure, PPPoE fonctionne comme PPP : on configure une pseudo-interface réseau. Dans le cas de PPP, elle s'appelle `ppp` ; avec PPPoE, il s'agit de `pppoe`. Rien de surprenant.

Le lecteur attentif aura remarqué que sous NetBSD 1.6, **`ifconfig -a`** ne liste pas d'interface `pppoe`. C'est normal : c'est une pseudo-interface clonable, que l'on peut créer ou détruire à volonté.

Les pseudo-interfaces clonables sont assez répandues dans les Unix les plus récents. Sous FreeBSD et NetBSD, on peut en obtenir la liste avec la commande **`ifconfig -C`** :

```
$ ifconfig -C
bridge vlan gif gre tun pppoe
```

Sous NetBSD, PPPoE se configure en plusieurs étapes. L'exemple suivant est extrait de la page **`man`** de **`pppoectl`**, dont la lecture est recommandée.

```
# ifconfig ne0 up ❶
# ifconfig pppoe0 create ❷
# pppoectl -e ne0 pppoe0 ❸
# pppoectl pppoe0 myauthproto=pap myauthname='toto' \
> myauthsecret='q' hisauthproto=none ❹
# ifconfig pppoe0 0.0.0.0 0.0.0.1 up ❺
```

❶ Activation de l'interface Ethernet `ne0`, où est connecté le modem ADSL.

❷ Création de la pseudo-interface clonable `pppoe0`.

❸ Liaison de `pppoe0` à l'interface `ne0`.

❹ Configuration de l'authentification, avec le nom d'utilisateur **`toto`** et le mot de passe **`q`**. Le **** (*backslash*, ou anti-barre oblique) permet en shell de continuer une ligne de commande sur la ligne suivante. Cela se remarque souvent au > affiché par la machine en début de ligne supplémentaire.

❺ Activation de l'interface `pppoe0`, qui va démarrer l'authentification. Les deux adresses sont factices ; le serveur en fournira de toutes façons d'autres.

ATTENTION **Modems USB**

Les systèmes Unix libres reconnaissent mal les modems ADSL USB, en général par manque de documentation sur ces équipements. La situation n'est pas forcément désespérée. Par exemple, pour son modem Speedtouch USB, Alcatel fournit un squelette de pilote sous forme binaire, à l'origine prévu pour Linux/i386. Il est aussi exploitable sur les PC équipés de systèmes BSD.
Dans le cas général, seuls les modems Ethernet fonctionnent à coup sûr. Pour les autres, renseignez-vous, mais la situation est assez peu favorable.

PLUS LOIN **D'autres usages de pppd**

`pppd` permet de se connecter à Internet par modem RTC. Configuré correctement, il peut aussi être utilisé pour transformer une machine Unix en serveur d'accès distant, en jouant le rôle du serveur.
Autre utilisation encore : **`pppd`** peut servir à monter un lien IP entre deux machines reliées par un câble série. Cela est très utile pour relier momentanément une machine dont la carte réseau ne fonctionne pas, pour par exemple télécharger un nouveau noyau capable de la gérer. On va exécuter la commande suivante sur la machine jouant le rôle du serveur :

```
# pppd tty00 115300 local noauth silent persist 192.168.1.2:192.168.1.3
```

Puis sur la machine jouant le rôle du client :

```
# pppd tty00 115300 local noauth defaultroute
```

Enfin, on peut utiliser **`pppd`** pour monter un réseau privé virtuel (VPN) par un lien PPP dans une connexion SSH. Une telle configuration sera abordée à la fin du chapitre 10.

CULTURE **À propos de câble et d'ADSL**

Les liaisons câblées et ADSL se déclinent sous la forme de différentes offres, variant de 64 Kb/s à 1 Mb/s. Le débit montant et le débit descendant sont souvent différents (le « A » d'« ADSL » ou de sa traduction française « RNA » signifie « asymétrique »), le débit descendant étant largement supérieur.
À titre de comparaison, les liaisons téléphoniques classiques (RTC) plafonnent à 56 Kb/s. La téléphonie numérique (Réseau Numérique à Intégration de Service, RNIS ; on dit *Integrated Service Digital Network* ou ISDN en anglais) permet elle des débits de 64 Kb/s ou 128 Kb/s en utilisant deux canaux à la fois. Le gros problème de ces liaisons n'est pas tant le débit que leur facturation en France : contrairement au câble et à l'ADSL, on paie à la minute et non pas au forfait.

ALTERNATIVE **ppp et pppd**

Disponibles sur les trois BSD, **ppp** et **pppd** implémentent tous les deux le protocole PPP. **pppd** s'appuie sur des couches PPP présentes dans le noyau, alors que **ppp** implémente la totalité de PPP hors du noyau.

EN CAS DE COUP DUR **Si cela ne fonctionne pas**

En cas d'échec, /var/log/messages est encore l'endroit à examiner pour avoir des pistes sur la cause du problème.
Si le serveur d'authentification semble ne pas répondre, vérifiez en premier lieu que la carte Ethernet fonctionne correctement. **tcpdump** est aussi très utile : il permet de contrôler les échanges entre la machine et le serveur d'authentification. Nous avons abordé ces problèmes de diagnostics dans la section consacrée à la configuration d'une connexion via un réseau Ethernet.

Configuration de PPPoE sous FreeBSD

Sous FreeBSD, on utilise les pseudo-interfaces `tun`, qui servent à construire des tunnels (cette notion sera abordée dans le détail au chapitre 9).

La marche à suivre comporte un peu plus de préliminaires que sous NetBSD. Tout d'abord, il faut charger dans le noyau tous les modules nécessaires. Ils sont au nombre de quatre : netgraph, ng_socket, ng_ether, et ng_pppoe. On ajoute donc les lignes suivantes dans /boot/loader.conf :

```
ng_socket_load="YES"
ng_ether_load="YES"
ng_pppoe_load="YES"
```

Pour charger les modules, le plus simple est de redémarrer (on peut aussi les charger à la main avec **lkdload**). **ppp** est le programme utilisé pour établir la connexion PPPoE. On le configure en ajoutant une entrée dans /etc/ppp/ppp.conf, que nous baptisons ici **adsl** :

```
adsl:
 set device PPPoE:ne0:
 set authname toto
 set authkey q
 set timeout 0
 set mtu 1492
 set mru 1492
 set speed sync
 enable lqr
 set lqrperiod 5
 set reconnect 5 10000
 set redial 5 10000
 disable acfcomp protocomp
 deny acfcomp
 set log Phase Chat IPCP CCP tun
 set ifaddr 0/0 0/0
 add 0 0 HISADDR
 set dial
```

Le fichier de configuration ainsi préparé, il ne reste plus qu'à invoquer **ppp** en indiquant le nom de la configuration que nous venons d'entrer dans /etc/ppp/ppp.conf :

```
# ppp adsl
```

L'exécution automatique au démarrage s'obtient en ajoutant quelques lignes au fichier /etc/rc.conf :

```
ppp_enable="YES"
ppp_mode="background"
ppp_profile="adsl"
```

Résolution de noms

Beaucoup de services utilisent des noms de machines, qu'ils tentent de résoudre en adresses IP. La résolution de noms sera la touche finale de notre configuration du réseau.

Comme on l'a vu précédemment, la résolution des noms de machines peut se faire via le DNS, si on dispose d'un tel serveur sur le réseau. Tout se configure dans le fichier /etc/resolv.conf, par exemple comme ceci :

```
nameserver 10.0.13.1
search local.net
```

On indique ici d'utiliser l'IP 10.0.13.1 comme serveur de noms, et d'ajouter le domaine `local.net` en cas de nom de machine non complété par son nom de domaine.

On peut encore résoudre les noms en indiquant dans le fichier /etc/hosts les associations entre noms et adresses IP. Cela permet de disposer d'une résolution de noms pour les machines critiques au démarrage des services, même en l'absence de DNS. Il n'est pas obligatoire de s'astreindre à des machines locales : on peut préciser des machines lointaines. Exemple de fichier /etc/hosts :

```
127.0.0.1       localhost
10.0.0.10       disco.local.net         disco
10.0.0.1        gw-ext.local.net        gw-ext
10.0.13.1       dns.local.net           dns
204.152.184.75  ftp.netbsd.org
```

Le nom de la machine elle-même se configure pour sa part à la ligne de commande en utilisant **`hostname`**. La variable `hostname` du fichier /etc/rc.conf assure la configuration automatique au démarrage :

```
hostname="spoutnik"
```

PIÈGE À C... **Inclure tout Internet dans /etc/hosts**

/etc/hosts est très pratique, mais il ne se met pas à jour automatiquement, contrairement au DNS. Chacune de ses entrées est donc susceptible de devenir obsolète. Si par exemple, dans le cas du /etc/hosts ci-contre, ftp.netbsd.org change d'adresse IP (comme suite à un déménagement), vous risquez de ne plus pouvoir le contacter sans comprendre pourquoi : vous utiliserez l'ancienne adresse précisée dans le fichier /etc/hosts. Pensez à ce fichier lors de la survenue d'un tel problème.

ASTUCE **Empêcher les résolutions DNS**

Un certain nombre de commandes de diagnostic et de configuration du réseau tentent de résoudre des noms : **`ping`**, **`route`**, **`netstat`**, **`traceroute`**. En cas de problème (DNS indisponible), l'option **`-n`** sera d'une aide précieuse : elle évite de devoir attendre plusieurs secondes que chaque requête DNS échoue (*timeout*).

L'interface graphique

Certaines machines disposent d'un mode graphique : c'est par exemple le cas sur les Macintosh et les PC sous Windows, ainsi que sur un certain nombre de stations de travail. S'il n'est pas utile sur les serveurs, le mode graphique est le point de passage obligatoire pour monter des solutions de bureautique sous Unix, ou simplement pour construire une station d'administration agréable à utiliser. Voyons comment configurer l'environnement graphique.

L'environnement *X Window System*

La grande majorité des systèmes Unix utilisent le système de fenêtrage X (*X Window System*) pour l'affichage en mode graphique. Il est organisé en mode client/serveur : on parle de serveurs X et de clients X.

Le serveur X gère un terminal graphique. Les clients X s'y connectent pour lui ordonner des opérations d'affichage ou collecter les informations en provenance du clavier et de la souris. Tout programme fonctionnant sous interface graphique est un client X : logiciel de dessin, traitement de texte, navigateur Web, ou éditeur de texte.

Les clients X peuvent travailler avec un serveur X local ou utiliser le réseau pour se connecter à un serveur distant. Il est ainsi possible d'exécuter des programmes sur une machine mais de les faire s'afficher sur une autre. Ceci permet d'utiliser des programmes en mode graphique sur une machine dénuée de terminal graphique.

SUR LES AUTRES UNIX **X Window System**

Certains systèmes n'utilisent pas *X Window System* pour leur affichage. NeXTStep et son descendant OpenStep utilisent DisplayPostScript ; MacOS X utilise Quartz, un système d'affichage basé sur PDF.
Ces systèmes sont similaires, quoique incompatibles, avec *X Window System*. Ils reposent sur le modèle client/serveur mais utilisent des protocoles et des interfaces de programmation différents. Ainsi, Quartz dispose de son propre serveur d'affichage, nommé **`WindowServer`**.

VOCABULAIRE **Terminal X**

Un terminal X est un terminal graphique piloté par un serveur X. Il comporte un écran, un clavier, et souvent une souris.
Il est possible de transformer un poste de travail Windows ou MacOS en terminal X en y employant un serveur X. On trouve aussi des terminaux X, ordinateurs simplifiés, sans disque ni lecteur de disquette ou CD-ROM. Leur seule fonction consiste à faire fonctionner un serveur X pour travailler à distance sur une machine Unix.

CULTURE **XFree86**

Les projets BSD et GNU/Linux partagent le même serveur X : celui du projet XFree86. Toute question sur *X Window System* pour BSD pourra donc trouver sa réponse dans la documentation écrite pour GNU/Linux.

Il existe quelques clients X particuliers : les gestionnaires de terminaux (*display manager*) et les gestionnaires de fenêtres (*window manager*). Les premiers sont chargés d'afficher une invite de connexion sur les terminaux dont ils ont la charge, puis d'exécuter le gestionnaire de fenêtres de tout utilisateur correctement authentifié.

Quant au gestionnaire de fenêtres, il initialise la session de l'utilisateur et pilote son environnement graphique. C'est lui qui donne une apparence et un comportement particuliers à l'interface utilisateur. Il gère notamment les bords des fenêtres, leur barre, leurs boutons, les menus de fond d'écran, les éventuelles barres de menus, et les *docks*, ou barres des tâches.

Configuration du serveur X

EN BREF **Configurer X**

La configuration de X pour les gens pressés : tentez un **`XFree86 -configure`**. Si cela ne fonctionne pas, essayez **`xf86config`**. Si vous n'obtenez toujours rien, vérifiez que la carte est bien prise en charge par votre version de XFree86. Si cela n'est pas le cas, mettez à jour XFree86 ou changez de carte.

Avant de pouvoir utiliser l'interface graphique, il faut configurer le serveur X. Cette opération, très rapide quand l'auto-configuration fonctionne, peut tourner au marathon dans le cas contraire. Les difficultés apparaissent avec les cartes vidéo ISA très anciennes, dont le serveur X sera incapable d'auto-détecter les caractéristiques. Quant aux cartes trop récentes, elles ne disposeront pas encore de pilote dans le cadre du projet XFree86.

À l'heure où sont écrites ces lignes, XFree86 en est à sa version 4. Elle comprend un unique serveur X : /usr/X11R6/bin/XFree86, capable de charger des pilotes modulaires pour gérer toutes les cartes vidéo prises en charge. Pour savoir quel pilote utiliser, le serveur **`XFree86`** utilise un fichier de configuration : XF86Config, placé sous /usr/X11R6/lib/X11 ou /etc/X11. En voici un exemple entrecoupé d'explications quant à ses différentes sections :

PLUS LOIN **XFree86 version 3**

Les explications données ici concernent surtout XFree86 version 4. Sur la version 3, il existe différents serveurs X, tous prévus pour une carte vidéo donnée. Ainsi, les PowerMacintosh ont pu utiliser le serveur **`Xmacppc`**, qui ne nécessitait aucune configuration – mais qui n'utilisait aucune capacité d'accélération du matériel.
La plupart des serveurs X de XFree86 version 3 écrits pour les cartes vidéo des PC utilisent un fichier de configuration XF86Config. Celui-ci peut être écrit à la main ou à l'aide du programme **`xf86config`**. La commande **`SuperProbe`** peut permettre de retrouver certaines caractéristiques de la carte vidéo réclamées par **`xf86config`**.

```
Section "Files"
    RgbPath     "/usr/X11R6/lib/X11/rgb"
    FontPath    "/usr/X11R6/lib/X11/fonts/local/"
    FontPath    "/usr/X11R6/lib/X11/fonts/misc/"
    FontPath    "/usr/X11R6/lib/X11/fonts/75dpi/:unscaled"
    FontPath    "/usr/X11R6/lib/X11/fonts/100dpi/:unscaled"
    FontPath    "/usr/X11R6/lib/X11/fonts/Type1/"
    FontPath    "/usr/X11R6/lib/X11/fonts/TrueType/"
    FontPath    "/usr/X11R6/lib/X11/fonts/TTF/"
    FontPath    "/usr/X11R6/lib/X11/fonts/Speedo/"
    FontPath    "/usr/X11R6/lib/X11/fonts/75dpi/"
    FontPath    "/usr/X11R6/lib/X11/fonts/100dpi/"
#   FontPath    "unix/:7101" ❶
    ModulePath "/usr/X11R6/lib/modules"
EndSection
```

La section `Files` indique l'emplacement de divers fichiers : descriptions des couleurs, polices de caractères, et répertoires où sont installés les modules du serveur X. Sauf installation particulière, il n'est pas utile de s'écarter des réglages par défaut.

❶ Ligne indiquant l'utilisation du serveur de polices **`xfs`**.

PLUS LOIN **`xfs`**

`xfs` est un serveur de polices de caractères. Il permet de centraliser l'installation des polices, et donc de gagner de l'espace disque.
Il existe également un serveur dédié aux polices TrueType : **`xfstt`**.

```
Section "Module"
    Load       "dbe"
    Load       "type1"
    Load       "freetype"
    Load       "glx"
    SubSection "extmod"
        Option  "omit XFree86-DGA"
    EndSubSection
EndSection
```

Dans cette section, on indique le chargement de quelques extensions : *double buffering* (tampon double), gestion des polices de caractères de type 1 et FreeType, et extensions OpenGL. Les modules d'extensions sont installés dans /usr/X11R6/lib/modules/extensions, mais leur documentation est assez éparpillée.

```
Section "ServerFlags"
#    Option "NoTrapSignals"
#    Option "DontZap"
#    Option "DontZoom"
#    Option "DisableVidModeExtension"
#    Option "AllowNonLocalXvidtune"
#    Option "DisableModInDev"
#    Option "AllowNonLocalModInDev"
EndSection
```

Cette section permet de configurer quelques options du serveur X, comme la possibilité de le tuer avec la combinaison de touches **Control**+**Alt**+**Effacement**. Ces options sont documentées dans la page **man** `XF86Config(5)`

```
Section "InputDevice"
    Identifier  "Keyboard1"
    Driver      "Keyboard"
    Option      "AutoRepeat" "200 10"
    Option      "XkbRules"   "xfree86"
    Option      "XkbModel"   "pc105"
    Option      "XkbLayout"  "fr"
EndSection
```

Description d'un clavier. Cette section porte un champ `Identifier`, qui permet de la référencer plus loin dans le fichier. Tout identifiant est valable, tant qu'il reste unique dans le fichier.

Si votre machine a plusieurs claviers, c'est celui de la console qui sera utilisé, à moins d'indiquer un fichier spécial du /dev par une ligne `Option "Device"`. C'est aussi dans cette section que l'on configure le temps de répétition des touches, ou le type de clavier (ici 105 touches français).

```
Section "InputDevice"
    Identifier  "Mouse1"
    Driver      "mouse"
    Option      "Protocol"          "wsmouse"
    Option      "Device"            "/dev/wsmouse"
    Option      "Buttons"           "3"
    Option      "Emulate3Buttons"   ❶
    Option      "Resolution"        "200"
    Option      "ZAxisMapping"      "4 5" ❷
EndSection
```

Configuration d'une souris. La recherche du protocole et fichier spécial à indiquer dans cette section est très irritante. NetBSD et OpenBSD apportent une réponse élégante à ce problème, en proposant un pilote de souris universel : toutes les souris sont vues comme étant de type `wsmouse` et connectées sur un fichier spécial /dev/wsmouse.

Comme les sections de description de clavier, celles concernant les souris doivent porter un identifiant unique pour pouvoir être référencées plus loin dans le fichier.

❶ Cette option permet de vivre au-dessus de ses moyens, en émulant trois boutons sur une souris à deux boutons. On obtient le troisième bouton en cliquant sur les deux boutons à la fois.

❷ Configuration de la molette pour les souris qui en sont équipées.

CAS PARTICULIER `wsmouse` **et les souris série**

Dans une installation raisonnablement moderne, la souris est PS/2, USB, ou ADB. `wsmouse` n'a aucun mal à gérer ces souris de façon transparente car elles sont soit prévues pour être auto-détectées (souris USB et ADB), soit affectées à un port ne pouvant servir qu'à une souris (souris PS/2).

Les antiques souris série ne jouissent pas d'une situation aussi favorable : `wsmouse` ne peut pas les différencier de façon fiable des autres périphériques susceptibles d'être connectés au port série. L'administrateur doit donc y mettre sa patte et exécuter le *daemon* **moused** pour que la souris soit prise en charge en tant que souris `wsmouse`.

```
Section "Monitor"
    Identifier  "My Monitor"
    HorizSync   31.5-57.0
    VertRefresh 50-90
EndSection
```

Cette section décrit les caractéristiques d'un écran. On y indique les intervalles de fréquences supportées par l'écran : balayage horizontal (en kHz) et rafraîchissement vertical (en Hz). Si vous êtes équipé d'une carte vidéo et d'un moniteur récents, gérant tous deux le DDC (*Display Data Channel* : canal de données d'affichage), le serveur X peut auto-détecter les capacités de l'écran, et ces informations deviennent alors facultatives. Sinon, la documentation de votre moniteur devrait vous les fournir. Par défaut, vous pouvez toujours essayer l'affichage VGA standard : 31,5 kHz et 60 Hz.

Pour référence ultérieure dans le fichier, cette section doit également porter un identifiant unique.

```
Section "Device" ❶
    Identifier  "ATI"
    Driver      "radeon"
EndSection

Section "Device" ❷
    Identifier  "VGA"
    VendorName  "Unknown"
    BoardName   "Unknown"
    Driver      "vga"
EndSection
```

Les sections `Device` doivent également porter un identifiant unique. On les utilise pour indiquer au serveur X quelles cartes vidéo on souhaite exploiter, et avec quels pilotes.

❶ Une carte ATI, gérée par le pilote radeon. La liste des cartes vidéo gérées par XFree86 version 4 et le nom des pilotes les prenant en charge se trouve sur le Web : http://www.xfree86.org/current/Status.html. Les pilotes sont installés sous forme de modules dans /usr/X11R6/lib/modules/drivers.

❷ Cette section permet de configurer la carte vidéo en mode VGA. Elle est ici à titre d'exemple est n'est pas utilisée dans la suite de ce fichier. Toute carte capable de gérer le standard VGA peut être configurée ainsi, même si XFree86 n'a aucun pilote pour la gérer. L'affichage sera en revanche limité à 640 par 480 pixels en 16 couleurs (soit 4 bits par pixel).

```
Section "Screen"
    Identifier  "Screen1"
    Device      "ATI" ❶
    Monitor     "My Monitor"
    DefaultDepth 24 ❷

    Subsection "Display"
        Depth       24
        Modes       "1280x1024" "1024x768"
        ViewPort    0 0 ❸
        Virtual     1280 1024
    EndSubsection

    Subsection "Display"
        Depth       8
        Modes       "1280x1024" "1024x768"
        ViewPort    0 0
        Virtual     1280 1024
    EndSubsection
EndSection
```

EN CAS DE COUP DUR **Désactiver le DDC**

Le DDC, censé simplifier la vie de l'usager en autorisant l'auto-configuration, est parfois source de dysfonctionnements. Si votre serveur X plante au démarrage, cela peut être à cause du DDC. Pour désactiver celui-ci, vous pouvez ajouter la ligne suivante dans la section `Device` correspondante :

```
    Option    "NoDDC"
```

On peut aussi tenter de désactiver sélectivement les versions 1 ou 2 du DDC, en indiquant `"NoDCC1"` ou `"NoDCC2"`.

PLUS LOIN **Cartes vidéo multiples**

Lorsque la machine dispose de plusieurs cartes vidéo, ou si le serveur X est incapable de trouver lui-même la carte, on peut indiquer dans une section `Device` le bus et la position où le matériel est situé. La page **man** de XF86Config décrit cette possibilité, ainsi que d'autres réglages qui peuvent être faits explicitement si le serveur X n'est pas capable de les trouver seul.

Avant-dernière section du fichier, qui doit toujours porter un identifiant unique, la section `Screen` lie entre elles une carte vidéo et une spécification d'écran, en utilisant les identifiants uniques employés dans chacune des sections précédentes.

❶ C'est en modifiant la ligne `Device` que l'on peut par exemple choisir entre le pilote radeon, défini dans la section `Device` nommée `"ATI"`, ou le pilote générique VGA, défini dans la section `Device "VGA"`

❷ On définit ici la profondeur d'écran à utiliser par défaut. Chaque possibilité est ensuite traitée par une sous-section `Display`, où sont indiquées les résolutions à employer, la résolution par défaut étant donnée en premier. Le changement (cyclique) de résolution se fait par les combinaison de touches **`Control`**+**`Alt`**+**`+`** et **`Control`**+**`Alt`**+**`−`**.

❸ Ces deux paramètres permettent de vivre au-dessus de ses moyens en utilisant un bureau plus grand que l'écran. `ViewPort` indique le placement initial de l'écran sur le bureau virtuel, et `Virtual` indique la taille du bureau virtuel.

`Virtual` sert aussi à obtenir un bureau de la taille de l'écran lorsque l'autoconfiguration du serveur X a décidé que le bureau serait virtuel. Il suffit pour cela d'indiquer des dimensions identiques à celles de la ligne `Modes`.

B.A.-BA **Profondeur d'écran**

Par ce terme on désigne le nombre de bits utilisés pour encoder la couleur d'un pixel. Une profondeur d'écran de 1 bit correspond à un affichage monochrome. 8 bits donnent 256 couleurs, 16 bits plusieurs milliers, et 24 bits plusieurs millions.

```
Section "ServerLayout"
    Identifier  "Layout1"
    Screen      "Screen1"
    InputDevice "Mouse1"
    InputDevice "Keyboard1"
EndSection
```

La dernière section est facultative lorsque l'on n'a pas défini de multiples claviers, souris ou écrans. Elle associe les périphériques d'entrée et d'affichage en utilisant les identifiants uniques employés pour les désigner.

Le fichier XF86Config peut bien évidemment être rédigé à la main, mais c'est un exercice assez pénible. On préfère en général le faire fabriquer par un programme de configuration. Si l'on dispose de matériel récent, on peut tenter l'autoconfiguration complète, avec la commande **`XFree86 -configure`**. Le serveur

CULTURE **Modes d'affichage**

Le VGA (*Video Graphics Array*) a été introduit en 1987 par IBM. Il propose un affichage de 640 par 480 pixels avec 16 couleurs, ou bien 320 par 200 en 256 couleurs. C'est devenu le standard d'affichage minimum géré par tous les compatibles PC, remplaçant les anciens standards, également proposés par IBM : affichage en mode texte, modes CGA (*Color Graphics Adapter*) et EGA (*Enhanced Graphics Adapter*).
Aucune spécification n'a encore réussi à détrôner le VGA du rôle de standard minimum, que ce soit le XGA (*eXtended Graphics Array*) d'IBM, ou plus récemment les spécifications VESA (*Video Electronics Standards Association*) : SVGA (Super VGA), SXGA (Super XGA), et UXGA (Ultra XGA).
Le tableau ci-contre récapitule les caractéristiques de ces différents standards.

Standard	Résolution	Couleurs
CGA	320 par 200	4
EGA	640 par 350	16
VGA	640 par 480	16
	320 par 200	256
XGA	800 par 600	millions
	1024 par 768	65536
SVGA	800 par 600	millions
SXGA	1280 par 1024	millions
UXGA	1600 par 1200	millions

X tentera alors de détecter tout le matériel pour produire un fichier de configuration XF86Config.new. Parfois, l'auto-détection ne fonctionne pas parfaitement et il faut retoucher le fichier de configuration, mais cette opération aura déjà épargné beaucoup d'efforts.

L'autre voie, c'est le programme **xf86config**. Ce programme posera un certain nombre de questions sur le système, puis produira un fichier de configuration que vous pourrez retoucher à la main.

Si la carte vidéo est introuvable dans les choix proposés par **xf86config**, c'est probablement qu'elle est trop récente pour cette version de XFree86. Voyez sur http://www.xfree86.org/current/Status.html si votre version de XFree86 dispose bien d'un pilote pour la carte. Dans le cas contraire, trois solutions s'offrent à vous : soit utiliser la carte en mode VGA – donc en 640 par 480 avec 16 couleurs ; soit mettre à jour XFree86 – si une nouvelle version sait gérer correctement cette carte ; soit changer de carte.

Pour mettre à jour XFree86, consultez le site Web http://www.xfree86.org. N'oubliez pas de vérifier que la carte vidéo est bien reconnue par la dernière version, et suivez les instructions données dans les notes d'installation.

Il arrive parfois que la carte vidéo soit correctement prise en charge par XFree86 mais qu'un problème dans le noyau empêche son exploitation correcte. Le plus souvent, ce genre de problème se manifeste avec des cartes vidéo AGP. La seule solution est alors la mise à jour du noyau (le plus souvent en version *-current*) – si les dernières versions gèrent correctement la carte, bien entendu. Le chapitre 13 traite de ce genre de mise à jour.

Passage en mode graphique

Une fois la configuration terminée, on peut tenter un passage en mode graphique. Quelle que soit la version de XFree86, quel que soit le nom du serveur X retenu, un lien a normalement été installé donnant accès à celui-ci depuis /usr/X11R6/bin/X. Pour passer en mode graphique, il suffit donc d'invoquer la commande **X**. Sur certaines architectures, seul root peut l'invoquer ; sur d'autres, il est accessible à tous.

Exécuter **X** à la main présente en général un intérêt limité au simple test. Cette opération fait passer la console de la machine en mode graphique : on obtient alors un écran gris (avec un motif pied-de-poule), contenant uniquement le pointeur de la souris. Faute de client X, on ne peut y rien faire ! Pour observer d'autres éléments, il faudrait invoquer des clients X comme **xterm**, ou un gestionnaire de fenêtres.

On peut invoquer les clients X un par un en ligne de commande, mais ce n'est guère pratique. Pour exécuter X à la demande, on utilise plutôt le script **startx**, qui invoque le serveur X et exécute le script shell .xinitrc du répertoire personnel de l'utilisateur – qui doit être exécutable. Ce fichier se contente en général de démarrer un gestionnaire de fenêtres. Exemple de fichier .xinitrc :

EN CAS DE COUP DUR **Impossible de se connecter sous X**

Les erreurs survenant pendant l'exécution du script .xsession sont consignées dans le fichier .xsession-errors du répertoire personnel de l'utilisateur. Quand le serveur X démarre sans que le gestionnaire de fenêtres ne s'affiche, c'est là qu'il faut chercher le message d'erreur exposant le problème.
Attention aux disques pleins et aux quotas dépassés : ces situations empêchent souvent de se connecter sous X mais ne produisent aucun message d'erreur, faute d'espace disque pour les enregistrer !
Si c'est le serveur X qui refuse de démarrer, on contrôlera ses journaux, en général consignés dans un fichier du répertoire /var/log. Si X est invoqué par le biais de **startx**, on peut collecter d'éventuels messages d'erreur comme suit : **startx 2>&1 > log** (ce dernier nom représentant le fichier journal utilisé).

PIÈGE À C... **Rester coincé en mode graphique**

Après avoir invoqué un serveur X sans aucun client, vous ne pourrez plus rien faire, à part déplacer le curseur de la souris. En particulier, on n'a pas toujours la possibilité de tuer le serveur pour reprendre la main.
Certains systèmes Unix proposent plusieurs terminaux virtuels sur la console : on passe de l'un à l'autre par une combinaison de touches, comme **Control**+**Alt**+**F1** à **Control**+**Alt**+**F12** sur PC. Le serveur X occupe un terminal virtuel, mais on peut obtenir une console texte dans un autre terminal virtuel pour le tuer.
Sans terminaux virtuels, on peut tenter de reprendre la main en se connectant par le réseau. Certains serveurs X mourront aussi lors de l'enfoncement d'une combinaison de touches particulière, telle que **Control**+**Alt**+**Effacement** sur PC employant GNU/Linux ou BSD.

CULTURE **xterm**

xterm est un client X qui affiche simplement une fenêtre contenant un terminal texte.

```
#!/bin/sh
/usr/X11R6/bin/twm
```

twm est le gestionnaire de fenêtres historique du projet XFree86, et devrait être présent sur tous les systèmes Unix utilisant XFree86. Il est bien entendu possible d'utiliser un autre gestionnaire de fenêtres présent sur le système. Sur les BSD, seul **twm** est installé par défaut, mais de nombreux autres gestionnaires de fenêtres sont disponibles par le système de paquetages, et nous verrons comment les installer au chapitre 11.

Les machines fonctionnant en permanence avec l'interface graphique utilisent un gestionnaire de terminal, dont le plus ancien s'appelle **xdm**. Tous ses fichiers de configuration sont en général placés dans le répertoire /usr/X11R6/lib/X11/xdm. **xdm** utilise le fichier Xservers pour déterminer comment invoquer le serveur X. Il affiche ensuite une invite de connexion et exécute le script .xsession du répertoire personnel de tout utilisateur convenablement authentifié. Ce script, qui invoquera le gestionnaire de fenêtres, est semblable au fichier .xinitrc.

On peut exécuter **xdm** à la main (en tant que root), ou le démarrer automatiquement avec la machine. Sur NetBSD, ceci se fait en ajoutant la ligne **xdm=YES** à /etc/rc.conf. Sur OpenBSD, il faut indiquer **xdm_flags=""**, et sur FreeBSD, il faut invoquer **xdm** depuis /etc/rc.local.

Pour mettre fin à l'interface graphique, c'est bien **xdm** et non pas le serveur X qu'il faut tuer : en effet, **xdm** relance systématiquement ce dernier s'il meurt.

xdm étant fruste et peu configurable, les projets d'environnement de bureau GNOME et KDE ont programmé leur propre gestionnaire de terminal : **kdm**, issu du projet KDE, trouve ses fichiers de configuration sous /usr/X11R6/share/kde/config/kdm. **gdm**, originaire du projet GNOME, stocke lui sa configuration sous /usr/X11R6/etc/gdm.

CULTURE **GNOME et KDE**

GNOME et KDE sont deux projets d'environnement de bureau intégré (on dit *desktop* en anglais) sous Unix. Ils proposent tous deux un gestionnaire de terminal, un gestionnaire de fenêtres, des applications d'administration du système, des applications de bureautique, des gestionnaires de fichiers... Le but est de monter une solution complète, cohérente, et intégrée. Malheureusement, ces environnements démarrent bien plus lentement qu'un simple gestionnaire de fenêtres comme WindowMaker ou Blackbox.

GNOME et KDE, très populaires, sont fournis en standard sur un certain nombre de distributions Linux. Pour les utiliser sur les systèmes BSD, il faut d'abord les installer (nous verrons cela au chapitre 11). GNOME sera ensuite invoqué par /usr/X11R6/bin/gnome-session dans .xsession. Dans le cas de KDE, il faudra exécuter /usr/X11R6/bin/startkde.

Certains Unix constructeur proposent le bureau CDE. C'est un logiciel propriétaire et commercial. On peut lui trouver un substitut dans le monde du logiciel libre : **xfce**.

Gestionnaires de fenêtres

Sur un BSD fraîchement installé, on ne trouvera que **twm**, qui est extrêmement simple. Si vous le trouvez trop fruste, vous pouvez en installer un autre. Un encadré présente une liste de gestionnaires de fenêtres disponibles en logiciel libre, avec photos d'écran, ce qui pourra guider votre choix.

Sur les Unix constructeur, vous trouverez aussi quelques gestionnaires de fenêtres propriétaires :

- **mwm**, *Motif Window Manager*, disponible sur de nombreux Unix.
- **4Dwm**, variante de **mwm** pour IRIX sur stations *Silicon Graphics* (SGI).
- **vuewm**, variante de **mwm** pour HPUX sur stations HP PA-RISC.
- **dtwm**, variante de **mwm** pour le bureau CDE, disponible sur de nombreux Unix.
- **olwm** et **olvwm**, *Open Look Window Manager*, qui remplacent souvent CDE sur Solaris.

ALTERNATIVE **Lequel choisir ?**

Le choix de l'environnement graphique est difficile : les possibilités abondent. Après la sélection d'un environnement de bureau, il faudra encore se fixer sur un gestionnaire de fenêtres.

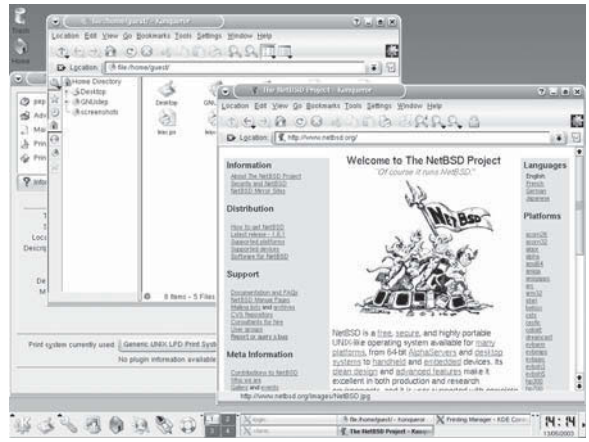
On peut très bien n'utiliser un environnement de bureau que partiellement : le gestionnaire de terminal de KDE cohabitera avec un gestionnaire de fenêtres indépendant, et des applications diverses et variées.

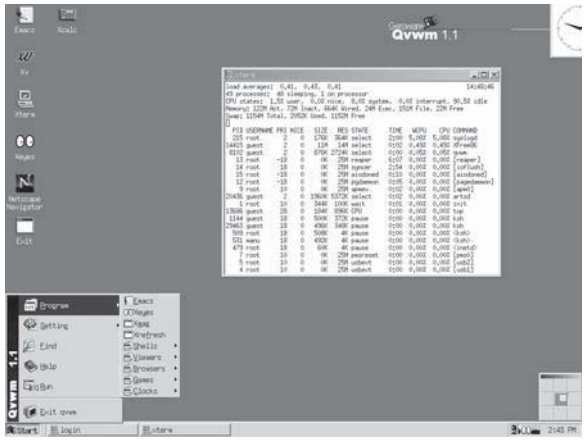
La préférence de l'auteur va au gestionnaire de terminaux **kdm**, assorti du gestionnaire de fenêtres **WindowMaker**, sans bureau intégré. Mais cela n'est qu'une affaire de goûts ; essayez plusieurs environnements pour vous faire une opinion.

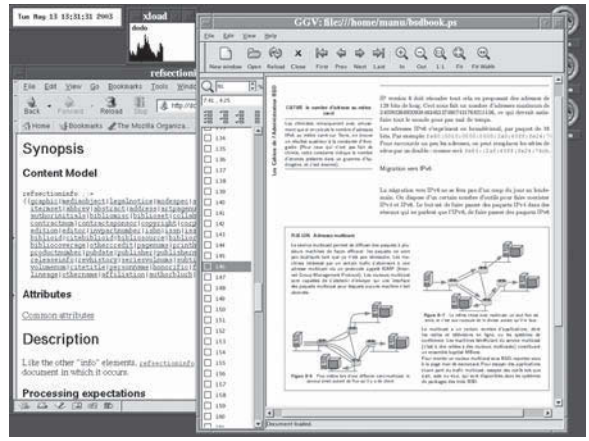
EN PRATIQUE **Quelques photos d'écran**

Voici quelques photos d'écran prises sur une machine NetBSD (elles seraient identiques sur FreeBSD, OpenBSD ou GNU/Linux). De gauche à droite et de haut en bas, on trouvera :

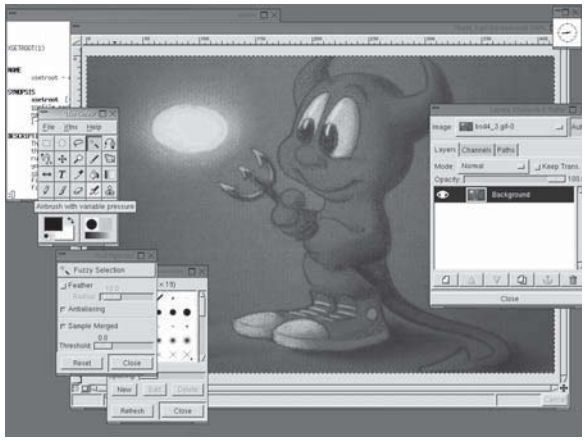
- Le très classique **`twm`**, présent partout, extrêmement fruste, mais aussi très léger. Côté applications, on a un **`xterm`**, le navigateur Mozilla, et **`xload`**, pour surveiller la charge de la machine.
- Le bureau KDE et son navigateur Konqueror, utilisable comme navigateur Web (au premier plan, connecté ici au site du projet NetBSD), ou comme gestionnaire de fichiers (à l'arrière-plan).
- Tous les goûts sont dans la nature, même le mauvais goût. Si vous avez du mal à décrocher de Windows, **`qvwm`** est fait pour vous. Sur ces images, on ne voit pas la cerise sur le gâteau : les icônes sont animées. C'est horrible.
- Pour un gestionnaire de fenêtres léger mais plus configurable que **`twm`**, essayez **`fvwm2`**. Côté applications, Ghostview, présentant la maquette du présent cahier au format PostScript. À l'arrière plan, Mozilla, consultant *DocBook : The definitive guide*, de Normal Walsh et Leonard Muellner (dialecte SGML/XML, utilisé pour écrire des documentations techniques, et en particulier ce cahier).

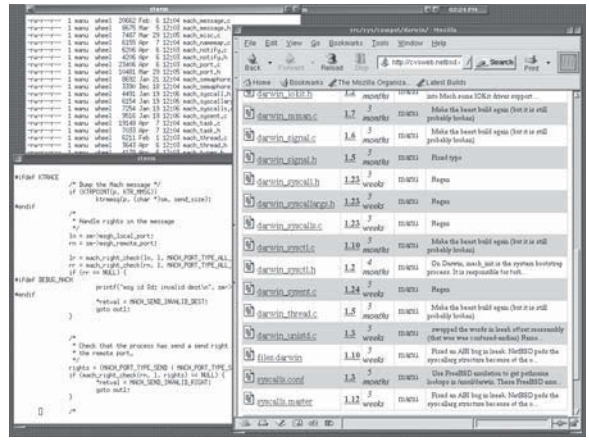


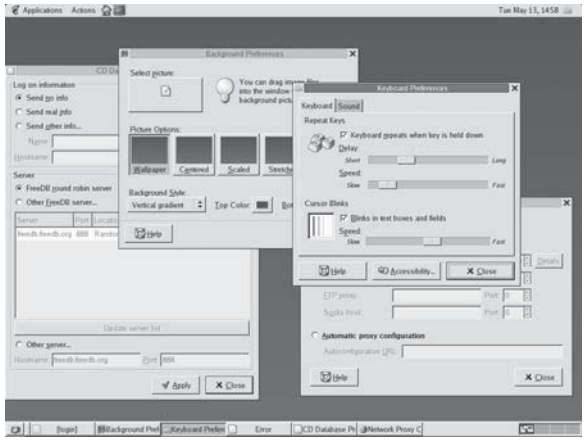


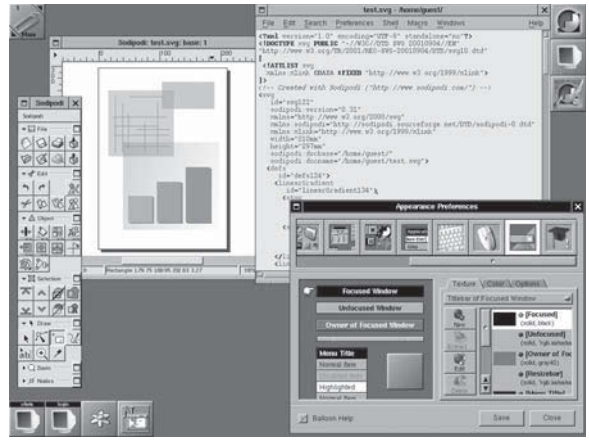


- Waimea, un autre gestionnaire de fenêtres léger, encadrant *The Gimp*, logiciel de retouche d'images.
- Autre gestionnaire léger : Blackbox. Côté applications, nous nous trouvons ici dans un environnement de développement. Au premier plan, Mozilla, qui affiche le CVSWeb du projet NetBSD, pour en suivre les développements à travers une interface Web. Derrière, on travaille sur la couche d'émulation du noyau Mach pour utiliser des binaires Darwin sous NetBSD/macppc.
- Le bureau GNOME et quelques-uns de ses tableaux de bord.
- Un très bon compromis de légèreté et d'élégance : WindowMaker. Rapide à charger et à utiliser, il dispose de tableaux de bord facilitant sa configuration. Nous voyons Sodipodi, logiciel de dessin vectoriel manipulant le format SVG (dialecte XML) et NEdit, éditeur de texte assez convivial, examinant ici le code SVG produit par Sodipodi.









Gestion des imprimantes

La gestion des imprimantes est encore un domaine où les différentes versions d'UNIX ont pu apporter leurs contributions. Il existe deux systèmes historiques : les systèmes LP (*Line Printer*) de UNIX System V, et LPR (*Line Printer Request*, parfois appelé LPD, pour *Line Printer Daemon*) de BSD. LPR, plus évolué, a d'ailleurs été adopté par quelques Unix basés sur Unix System V. S'y ajoutent deux systèmes plus récents et plutôt présents sous GNU/Linux : CUPS (*Common Unix Printing System*) et LPRng (*Line Printer Request new generation*).

Comme on pourrait s'y attendre, les systèmes BSD proposent LPR dans le système de base, alors que CUPS et LPRng sont disponibles dans les systèmes de paquetages.

ASTUCE **Imprimer sans le système d'impression**

On peut tout à fait imprimer en contournant le système d'impression standard. On y perd la gestion des files d'attente, mais c'est assez pratique pour déboguer un problème de communication avec l'imprimante. Ainsi, la commande suivante imprime la page **man** de **mv** sur une une imprimante PostScript connectée sur /dev/lpt0 : **groff -man -Tps /usr/share/man/man1/mv.1 > /dev/lpt0**.

Configuration d'imprimante locale avec LPR

Dans l'architecture LPR, les imprimantes sont configurées par le biais du fichier /etc/printcap. En voici un exemple, définissant une imprimante branchée sur le port parallèle et une imprimante branchée sur un port USB :

```
lp|Imprimante parallèle:\
       :mx#0:sh:lp=/dev/lpt0:sd=/var/spool/output/lp:lf=/var/log/lpd-errs:
lpusb:Imprimante USB\
       :mx#0:sh:lp=/dev/ulpt0:sd=/var/spool/output/lpusb:lf=/var/log/lpd-errs:
```

PLUS LOIN **Et les imprimantes réseau ?**

Le cas des imprimantes accessibles via le réseau sera examiné plus loin dans ce chapitre.

Chaque ligne correspond à une imprimante. Un \ (*antislash*) placé en fin de ligne permet de poursuivre la description à la ligne suivante.

Les lignes de description sont composées de champs séparés les uns des autres par le caractère **:** (deux points). Le premier champ indique les différents noms sous lesquels l'imprimante doit être connue, séparés par des | (barre verticale). Les champs suivants, sans ordre précis, renseignent des propriétés définies dans la page **man** printcap :

Propriété	Argument	Description
sh	Néant	Pas d'impression de page de garde.
mx#	Nombre	Taille maximum des documents acceptés, en blocs de 512 octets. Un argument nul ne limitera pas la taille.
lp=	Chemin	Chemin du fichier spécial donnant accès au port où est connectée l'imprimante.
sd=	Chemin	Chemin du répertoire de file d'attente d'impression.
lf=	Chemin	Chemin du fichier de journal des erreurs d'impression.

Le fichier spécial donnant accès à l'imprimante est le seul paramètre qui peut être difficile à retrouver. Sur les systèmes BSD, le port parallèle des PC est accessible par /dev/lpt0, et les imprimantes USB par /dev/ulpt0, /dev/ulpt1, *et cætera*. Si votre système est connecté à plusieurs imprimantes, la commande **dmesg** devrait vous être d'un grand secours : elle vous permettra de consulter les messages de détection du matériel émis par le noyau au démarrage, et dans lesquels vous retrouverez les noms de vos différentes imprimantes. Exemple :

```
lpt0 at isa0 port 0x378-0x37b irq 7
(...)
ulpt0 at uhub0 port 1 configuration 1 interface 0
ulpt0: Canon S630, rev 1.10/1.01, addr 2, iclass 7/1
ulpt0: using bi-directional mode
```

Une fois l'imprimante configurée, on peut invoquer le *daemon* **lpd**, dont le rôle est de gérer les files d'impressions. Ceci peut se faire en ligne de commande, ou automatiquement au démarrage, via le fichier /etc/rc.conf.

Vous pourrez alors utiliser les commandes suivantes, disponibles pour tout utilisateur : **lpr** pour effectuer une impression, **lpq** pour examiner l'état d'une file d'attente, et **lprm** pour supprimer une entrée de la file. Chacune de ces commandes utilise l'option **-P** pour d'indiquer le nom de l'imprimante à laquelle on s'intéresse. En son absence, c'est l'imprimante désignée par la variable d'environnement PRINTER qui sera utilisée, ou à défaut l'imprimante nommée lp (qui devra donc exister).

```
$ lpr -P lpusb rapport.ps
$ lpq -P lpusb
violette: sending to 10.0.12.9
Rank    Owner      Job  Files                                 Total Size
1st     manu       23   rapport.ps                            9598 bytes

$ lprm -P lpusb 23
dfA023violette dequeued
cfA023violette dequeued
$ lpq -P lpusb
no entries
$
```

En cas de dysfonctionnement, consultez le fichier de journal des erreurs d'impression dans /etc/printcap, assurez-vous de l'existence des files d'attentes d'impression, et vérifiez la bonne définition de l'imprimante dans /etc/printcap. Enfin, si l'imprimante imprime des horreurs sans rapport avec votre document, c'est que votre machine Unix lui parle une langue qu'elle ne comprend pas. La section suivante traite des problèmes de langages de description de pages.

Filtres d'impression et conversion au vol

Les premières imprimantes ne savaient imprimer que du texte. Le système d'impression se contentait de leur envoyer du texte brut, et les seuls malentendus pouvaient porter sur l'encodage (ASCII ou EBCDIC, par exemple) ou sur le format des retours de chariots (Macintosh, DOS ou Unix). Moyennant une éventuelle conversion au vol des fichiers, il était relativement facile de se faire comprendre de l'imprimante.

Les imprimantes modernes sont capables d'imprimer en mode graphique, ce qui complique la donne. L'imprimante parle un ou plusieurs langages de description de pages, tels que PostScript, PCL (*Printer Command Language*), PDF, ou un langage propriétaire spécifique à l'imprimante. Pour pouvoir imprimer sur ces machines, il faut leur envoyer les documents dans un format qu'elles comprennent.

Les applications Unix produisent généralement du PostScript, qui sera transmis au système d'impression par le biais de la commande **lpr** (ou **lp** sur System V).

SÉCURITÉ **lpd et le réseau**

Invoqué sans option particulière, le service **lpd** se comportera en serveur d'impression, et rendra vos imprimantes accessibles à travers le réseau via le protocole LPR (nous évoquerons plus loin les différents protocoles d'impression à travers le réseau). Pour éviter cela, passez-lui l'option **-s**.

SUR LES AUTRES UNIX **Les autres systèmes d'impression**

Dans le système LP, le service d'impression est assuré par **lpsched**, qui utilise en sous-main des agents d'impression dépendant de l'imprimante. Les commandes **lp**, **cancel** et **lpstatus** sont les équivalents System V des commandes **lpr**, **lprm** et **lpq** du système BSD. Dans CUPS, c'est **cupsd** qui assure le service d'impression, et la configuration se fait par le fichier cupsd.conf. CUPS fournit les commandes utilisateur de LP et LPR, ce qui permet de remplacer ces systèmes de façon transparente pour les usagers.
Enfin, LPRng reprend pour sa part l'interface d'administration de LPR. Le service est ainsi assuré par **lpd**, et se configure via /etc/printcap. Les commandes utilisateur sont celles de LPR, plus certaines commandes de LP.

PIÈGE À C... **Impression et retours de chariots**

Lorsque l'on envoie un fichier texte à une imprimante, on peut se heurter au problème des retours de chariots : le format habituel sous Unix consiste en un caractère LF (*Line Feed*, code ASCII 10). Si l'imprimante attend le retour de chariot au format DOS, c'est à dire CRLF (*Carriage Return* suivi de *Line Feed*, codes ASCII 13 et 10), on n'obtiendra pas le résultat escompté.
On peut régler ce problème en introduisant une conversion à la volée dans le système d'impression, ou en utilisant le programme **a2ps**, dont le rôle est de transformer des fichiers texte en PostScript. On contourne ainsi le problème des retours de chariot, mais l'imprimante doit être capable d'imprimer du PostScript. Ce programme est disponible dans les systèmes de paquetages des BSD, que nous examinerons au chapitre 11.

EN PRATIQUE **GhostScript**

GhostScript est disponible dans le système de paquetages des BSD.

Pour imprimer sur une imprimante PostScript, il n'y a donc rien de particulier à faire. En revanche, pour imprimer sur une imprimante PCL, il faudra convertir le PostScript en PCL.

LPR prévoit pour cela la configuration de filtres d'impression, par le biais de l'entrée `if=` du fichier /etc/printcap. On y indiquera le chemin d'un script faisant la conversion adéquate. Celle-ci est souvent réalisée par le programme GhostScript (**gs**), capable de transformer le PostScript en une foule de formats de sortie. Exemple d'entrée de /etc/printcap pour une imprimante Canon S630, qui est tout à fait incapable de comprendre le PostScript :

```
s630:Canon S630 USB:\
        :lp=/dev/ulpt0:sd=/var/spool/lpd:lf=/var/log/lpd-errs:\
        :mx#0:sh:if=/usr/local/bin/lpfilter-s630.sh:
```

Le filtre d'impression s'appelle /usr/local/bin/lpfilter-s630.sh, et voici son contenu :

```
#!/bin/sh
/usr/pkg/bin/gs @bj8pa06n.upp  -dSAFER -dBATCH -dQUIET -dNOPAUSE -q \
-sOutputFile=- -sPAPERSIZE=a4 - && exit 0
exit 2
```

L'option la plus importante des arguments de **gs** est le `@bj8pa06n.upp`, qui indique à GhostScript quel pilote utiliser. Il s'agit ici d'un pilote UniPrint (ou UPP). Les pilotes d'ancienne génération sont spécifiés par l'option **-sDEVICE** de **gs**. Exemple : **/usr/pkg/bin/gs -sDEVICE=cdj550**.

Lorsque l'on tente d'utiliser GhostScript pour travailler sur une imprimante non PostScript, le problème est surtout de savoir quel pilote utiliser. Le site http://www.linuxprinting.org est pour cela d'un grand secours. Principalement destiné à GNU/Linux, il se révèle être une mine d'informations sur l'impression sous Unix en général. On y trouve en particulier une base de données des imprimantes gérées par GhostScript et des pilotes correspondants. Ainsi, la Canon S630 déjà citée dispose d'une fiche détaillée, à l'adresse : http://www.linuxprinting.org/show_printer.cgi?recnum=Canon-S630

Signalons que tout cela n'est pas simple. En plus du pilote, il faut parfois fournir d'autres options particulières à **gs**. Autant il est facile d'imprimer depuis un système Unix sur une imprimante PostScript, autant l'impression sur les imprimantes non PostScript peut être délicate à mettre au point.

CULTURE **Encodages**

Les ordinateurs manipulent des octets, qui prennent des valeurs entre 0 et 255. Lorsque l'on rentre un texte dans la machine, il faut convenir d'un format d'encodage, qui consiste simplement à décider quels caractères correspondent à quelles valeurs. On parle aussi souvent de « jeu de caractères », ou *charset* en anglais.
Le code EBCDIC (*Extended Binary-Coded Decimal Interchange Code*) est par exemple largement utilisé par IBM sur ses grands systèmes. Les machines Unix utilisent traditionnellement l'ASCII (*American Standard Code for Information Interchange*), dont la table d'encodage est disponible dans la page **man** `ascii(7)`.
Le code ASCII est taillé pour l'échange de données en anglais ; il ne connaît pas les accents ou autres signes diacritiques. Plusieurs de ses extensions comblent cette lacune : les encodages historiques Mac Roman pour le Macintosh, Windows-1252 pour Windows, le plus récent encodage standard ISO-8859-1 (dit aussi ISO-Latin-1), et l'Unicode (UTF-8 et UTF-16).

CULTURE **Fichiers PPD**

Les fichiers PPD (*PostScript Printer Description* : fichier de description d'imprimante PostScript) décrivent dans le détail les fonctionnalités disponibles sur une imprimante : format des feuilles, polices disponibles, couleur... Ils sont à l'origine destinés aux systèmes Windows ou MacOS, mais le système d'impression CUPS a su rapidement les adopter pour tirer le meilleur parti des imprimante sous Unix.
Ces fichiers sont conçus pour les imprimantes PostScript, mais, à travers un filtre GhostScript, toutes les imprimantes deviennent capables d'imprimer du PostScript. PPD peut ainsi servir de format universel de description d'imprimantes.

Le programme **foomatic**, permet d'utiliser des fichiers PPD pour simplifier tout cela, en prenant en charge l'invocation de **gs** avec les options adéquates. La configuration d'une imprimante – qu'elle soit PostScript ou non – se limite alors à se procurer un fichier PPD et à indiquer son emplacement au système d'impression. Avec LPR, on l'indiquera simplement dans le /etc/printcap. Voici un remaniement de la configuration précédente en utilisant **foomatic** :

```
s630:Canon S630 USB: \
   :mx#0:sh:lp=/dev/ulpt0:sd=/var/spool/output/lpd:lf=/var/log/lpd-errs:\
   :if=/usr/local/bin/foomatic-rip:\
   :af=/usr/local/lib/Canon-S630-bj8pa06n.upp.ppd:
```

Le fichier PPD Canon-S630-bj8pa06n.upp.ppd est disponible sur le site http://www.linuxprinting.org, sur la page consacrée à la Canon S630. Ce site détaille aussi l'installation de **foomatic** et donne sa configuration avec les différents systèmes d'impression.

Imprimantes réseau

Traitons maintenant de l'impression sur des imprimantes accessibles à travers le réseau. C'est un sujet qui est devenu difficile avec le temps, non pas faute de standards, mais plutôt à cause de l'excès de normes régissant cette opération.

Le système LP ne fournit aucune possibilité d'impression à travers le réseau, ce qui a le mérite de simplifier les choses. Le système LPR a été le premier à proposer cette fonctionnalité, en introduisant le protocole LPR (souvent appelé protocole LPD). Ce dernier a joui d'une adoption très large, mais un certain nombre d'acteurs du marché de l'impression y ont ajouté leurs extensions propriétaires, et HP a introduit LaserJet, un protocole incompatible avec LPR. À cela il faut ajouter l'impression native des Macintosh sur AppleTalk, via le protocole PAP (*Printer Access Protocol*), l'impression native de Windows, via SMB (*Server Message Block*) sur NetBIOS ou TCP, et enfin une tentative de standardisation universelle, avec IPP (*Internet Printing Protocol*), qui est en réalité une extension de HTTP.

Le système LPR ne sait manipuler que le protocole éponyme. Les imprimantes réseau parlant celui-ci se configurent comme des imprimantes locales, dans le /etc/printcap. Voici un exemple de configuration d'imprimante réseau :

```
pepette|Apple Laserwriter 16/600 PS:mx#0:\
       :lp=:rm=10.0.12.9:rp=:sd=/var/spool/output/pepette: \
       :lf=/var/log/lpd-errs:
```

Le champ `lp=` reste ici vierge. Expliquons les nouveaux champs :

Propriété	Argument	Description
`rm=`	Adresse	Adresse de l'imprimante ou du serveur d'impression. On peut préciser un numéro de port : `9100@10.0.12.9`
`rp=`	Nom	Nom de file d'attente ou d'imprimante sur la machine distante. Utilisé surtout lorsqu'il s'agit d'un serveur d'impression gérant plusieurs imprimantes.

Si une fois l'imprimante réseau configurée vous ne pouvez pas imprimer, assurez-vous de l'absence de tout éventuel dispositif de filtrage (qui bloquerait vos tentatives d'accès) en exécutant par exemple un **`telnet`** sur le port où le serveur d'impression doit répondre :

```
$ telnet 10.0.12.9 515
Trying 10.0.12.9...
Connected to pepette.local.
Escape character is '^]'.
^]
telnet> quit
Connection closed.
$
```

Si un **`telnet`** sur le port 515 reste lettre morte, c'est que l'imprimante ne parle pas le protocole LPR. Sur certaines imprimantes, cette fonctionnalité peut être activée, mais sur d'autres, elle n'est pas disponible. Dans ce cas, il faudra vous tourner vers un autre système d'impression. CUPS et LPRng savent parler les protocoles LPR, JetDirect, et IPP. Pour accéder aux imprimantes ne parlant que les protocoles SMB ou PAP, il faudra vous tourner vers les logiciels Samba et Netatalk, qui sont bien évidemment disponibles dans les systèmes de paquetages.

B.A.-BA **Serveur d'impression**

Un serveur d'impression est une machine donnant accès à une ou plusieurs imprimantes à travers le réseau. Dans le cas d'une imprimante réseau, l'imprimante est son propre serveur d'impression, et celui-ci ne donne accès qu'à celle-là.

PLUS LOIN **Protocoles d'impression**

Voici un résumé des ports TCP utilisés par les protocoles d'impression sur IP les plus répandus :

Protocole	Port TCP
LPR	515
LaserJet	9100
SMB/NetBIOS	139
SMB/TCP	445
IPP	80

Serveur d'impression

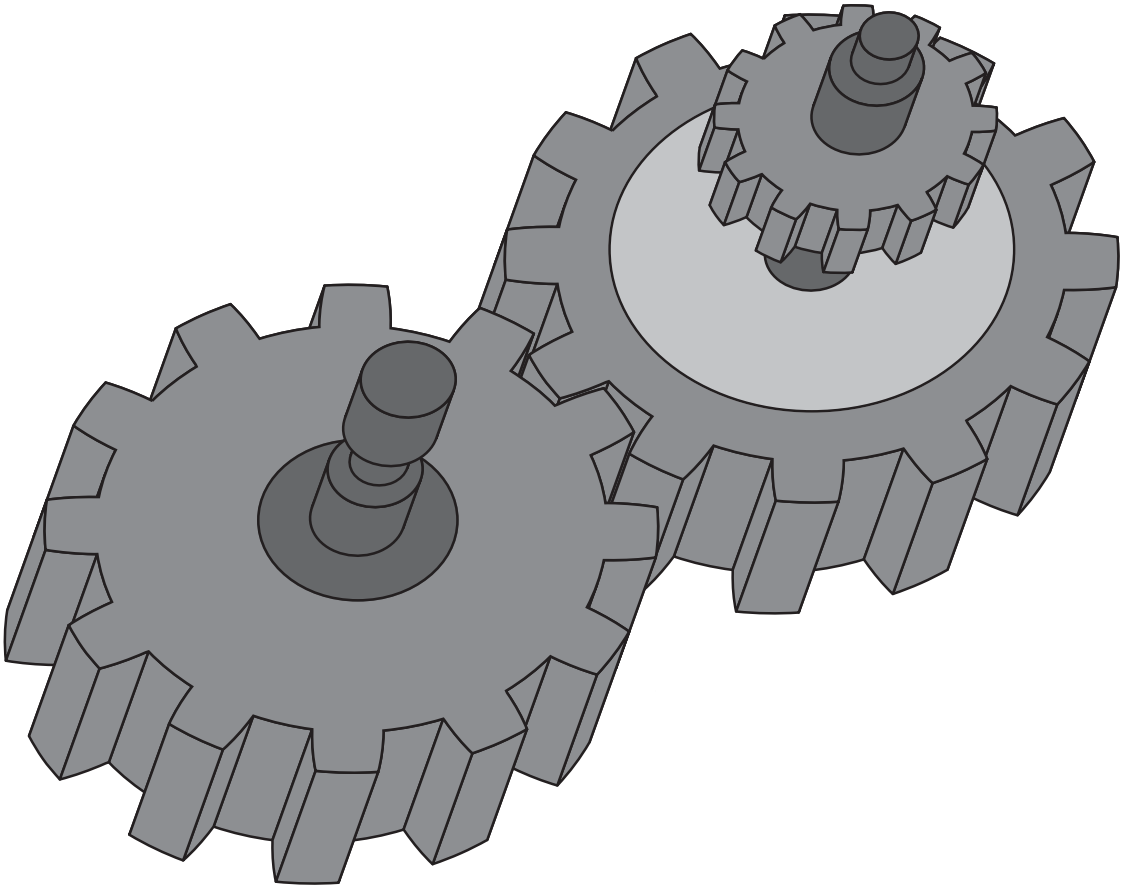
La mise en place d'un serveur d'impression LPR n'est pas très difficile : il suffit d'invoquer le *daemon* **`lpd`** sans l'option **`-s`**. Toutes les imprimantes accessibles localement deviendront alors accessibles par le réseau avec le protocole LPR.

En revanche, pour assurer un service d'impression pour les protocoles IPP, JetDirect, SMB ou PAP, il faut se tourner vers les logiciels CUPS, LPR-ng, Samba ou Netatalk.

EN PRATIQUE **Samba et Netatalk**

Samba et Netatalk transforment une machine en serveur de fichiers et d'impression respectivement pour Windows et pour MacOS. Ces deux logiciels savent également se comporter en clients : Samba gère le protocole SMB sur NetBIOS ou TCP, et Netatalk est capable de parler PAP sur AppleTalk et AFP (*Apple Filing Protocol*, utilisé par AppleShare) sur AppleTalk et TCP.
Ils disposent tous deux de fonctionnalités assez avancées. Ainsi, Samba peut transformer un serveur Unix en contrôleur de domaine Windows. Netatalk sait quant à lui se comporter en routeur AppleTalk entre deux segments d'Ethernet.

8



Services de base sous Unix

Les systèmes Unix sont tous peuplés par quelques *daemons* fondamentaux, que l'on retrouve partout. Certains assurent des services locaux, d'autres sont utilisés pour intégrer des machines Unix en réseau.

SOMMAIRE

MOTS-CLEFS

- Services
- syslogd
- inetd
- cron
- NFS
- NIS

Ce chapitre examine la configuration de *daemons* tels que **cron**, **syslogd**, **inetd** ou **nfsd**. Ces programmes sont des grands classiques du monde Unix, et leur mode de fonctionnement est assez standard d'un système Unix à l'autre.

Quelques services indispensables sous Unix

Certains services Unix sont suffisamment fondamentaux pour être activés dans l'installation par défaut. Nous allons examiner leurs usages et les bases de leur configuration.

Administration à distance : SSH

SSH est un protocole, décliné en deux versions, permettant de se connecter à distance à une machine. Tout le trafic est chiffré, ce qui empêche un indélicat écoutant la ligne de voler des informations sensibles (comme un mot de passe).

SSH a pour serveur **sshd**, qu'on peut au besoin invoquer directement depuis un shell de root. Pour qu'il démarre automatiquement avec la machine, tout dépend du système. Sur System V et GNU/Linux, il faut placer un lien dans un répertoire /etc/rcX.d ; pour NetBSD, il faut écrire **sshd=YES** dans /etc/rc.conf. Pour FreeBSD, il faut un **sshd_enable="YES"** dans le /etc/rc.conf. Sous OpenBSD, **sshd** est invoqué au démarrage dans la configuration par défaut.

L'implémentation la plus courante de SSH sous Unix est désormais OpenSSH, développé par l'équipe d'OpenBSD. C'est celle qu'on trouve dans les trois systèmes BSD libres et sous GNU/Linux. Ses fichiers de configuration ont beaucoup évolué, mais on peut espérer qu'ils ont désormais atteint une certaine stabilité. Ils se trouvent aujourd'hui dans /etc/ssh :

sshd_config	Configuration du serveur SSH.
ssh_config	Configuration du client SSH, commune à tous les utilisateurs du système.
ssh_host_key	Clef privée RSA pour le protocole SSH version 1.
ssh_host_key.pub	Clef publique RSA pour le protocole SSH version 1.
ssh_host_rsa_key	Clef privée RSA pour le protocole SSH version 2.
ssh_host_rsa_key.pub	Clef publique RSA pour le protocole SSH version 2.
ssh_host_dsa_key	Clef privée DSA pour le protocole SSH version 2.
ssh_host_dsa_key.pub	Clef publique DSA pour le protocole SSH version 2.

Avant de l'exécuter une première fois, il faut construire des clefs pour **sshd** : elles permettront aux clients SSH d'authentifier le serveur. Si elles changent, ces derniers signaleront à leurs utilisateurs que quelqu'un essaie peut-être d'usurper l'identité du serveur. C'est pourquoi il faut tenter de préserver ces clefs, même lors des mises à jour et des réinstallations. Si vous devez changer de clefs, avertissez les utilisateurs pour éviter de semer la panique.

EN PRATIQUE **Configuration du serveur SSH**

C'est dans le fichier /etc/ssh/sshd_config que l'on pourra par exemple désactiver les connections distantes de root via SSH, si cela n'est pas déjà le cas par défaut. C'est aussi le lieu précisant le chemin des clefs de la machine et toute une kyrielle d'options, documentées dans la page **man** de sshd_config.

On construit les clefs avec la commande **ssh-keygen**, en précisant l'emplacement où elles seront stockées. Les chemins doivent correspondre aux chemins précisés dans le fichier /etc/ssh/sshd_config :

```
# ssh-keygen -t rsa1 -b 1024 -f /etc/ssh/ssh_host_key -N ''
Generating public/private rsa1 key pair.
Your identification has been saved in /etc/ssh/ssh_host_key
Your public key has been saved in /etc/ssh/ssh_host_key.pub
The key fingerprint is:
14:c4:fb:78:90:55:8a:1e:2e:3a:3a:27:e1:b4:25:d2 root@violette
# ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key -N ''
Generating public/private rsa key pair.
Your identification has been saved in /etc/ssh/ssh_host_rsa_key
Your public key has been saved in /etc/ssh/ssh_host_rsa_key.pub
The key fingerprint is:
f7:24:5c:64:24:43:02:bc:d9:6d:47:3c:c6:68:39:c7 root@violette
# ssh-keygen -t dsa -f /etc/ssh/ssh_host_dsa_key -N ''
Generating public/private dsa key pair.
Your identification has been saved in /etc/ssh/ssh_host_dsa_key
Your public key has been saved in /etc/ssh/ssh_host_dsa_key.pub
The key fingerprint is:
12:7f:e4:c1:5b:b9:c7:f3:ba:89:4e:7e:26:6b:c3:64 root@violette
#
```

La première commande est utile pour la version 1 du protocole ; les deux autres sont propres à la version 2. Toutes ne seront nécessaires que pour invoquer **sshd** à la main, et seulement la première fois. En temps normal, les scripts d'installation ou d'initialisation s'occupent de tout.

SÉCURITÉ **Tenir à jour SSH**

Le fait que SSH chiffre les communications peut donner une fausse sensation de sécurité. Mais SSH n'est sûr qu'en l'absence d'erreur de programmation dans le client ou le serveur. Des bogues peuvent amener à une cryptographie plus faible que prévue, voire à des trous béants permettant par exemple de passer root à distance sans fournir d'authentification.

Comment éviter cela ? En se tenant au courant et en mettant à jour OpenSSH lorsque c'est nécessaire. Cette démarche, valable pour OpenSSH comme pour le reste du système, sera abordée dans le détail au chapitre 13.

PLUS LOIN **Connexion par clef RSA et DSA**

La description approfondie des algorithmes de cryptographie dépasse largement cadre de ce livre, mais on peut survoler la question.

RSA et DSA sont des méthodes de cryptographie utilisant des clefs asymétriques. Dans un tel système cryptographique, on trouve une clef publique, qui permet de chiffrer, et une clef privée, pour déchiffrer. Si un texte est chiffré avec la clef publique, seule la clef privée est capable de le déchiffrer. On peut donc utiliser ces clefs comme moyen d'authentification : un serveur disposant de ma clef publique m'envoie un texte chiffré ; je prouverai que je suis en possession de la clef privée correspondante en lui renvoyant le même texte déchiffré.

Comment utiliser des clefs asymétriques pour l'authentification avec SSH ? Il faut d'abord générer un couple de clefs avec la commande **ssh-keygen**. Cette dernière demande où enregistrer la clef privée, et enregistre la clef publique dans un fichier de même nom, suffixé par .pub. La clef privée se trouvera dans le répertoire .ssh/ du répertoire personnel de l'utilisateur.

On place ensuite la clef publique dans le fichier .ssh/authorized_keys de son répertoire personnel sur la machine distante (ou on l'y fait placer par l'administrateur). Ce fichier peut accueillir plusieurs clefs, une par ligne.

Enfin, il ne reste plus qu'à exécuter la commande **ssh** vers la machine distante, et on se connecte sans avoir à fournir de mot de passe : c'est la possession de la clef privée qui authentifie. L'authentification par clef asymétrique est très utile pour ouvrir un compte à une personne sans avoir à lui fournir un mot de passe par défaut, à modifier rapidement. L'utilisateur peut même ne pas avoir de mot de passe du tout (astérisque dans le champ adéquat du fichier de mots de passe).

En cas de problème, **ssh -vvv** permet d'avoir une sortie verbeuse côté client, et les journaux sont d'un grand secours côté serveur. Il s'agit souvent de problèmes de droits sur les fichiers de clefs : **sshd** refusera la connexion si ces droits sont trop permissifs. La clef privée ne doit être lisible que par l'utilisateur (mode 600). Le fichier authorized_keys doit être lisible par tous, mais modifiable par son seul propriétaire (mode 644).

Dans ce système, toute la sécurité repose sur la clef privée. Sa divulgation équivaut à la divulgation du mot de passe dans l'authentification usuelle, et permettra à un tiers d'usurper l'identité concernée. Si cette clef privée doit être entreposée à un emplacement non sûr (répertoire monté par NFS, machine où quelqu'un d'autre est root), il faudra la protéger par un mot de passe. Nécessaire pour déchiffrer la clef privée, ce mot de passe n'est pas envoyé au serveur.

PLUS LOIN **Entrée/sortie standard**

Les programmes Unix, écrits pour fonctionner dans un terminal, comptent trois flux d'entrée/sortie :

- L'entrée standard (*stdin*), sur laquelle le processus lit les données à traiter, tapées sur le terminal ;
- La sortie standard (*stdout*), utilisée pour afficher des messages ou les résultats d'un traitement ;
- Enfin, l'erreur standard (*stderr*), où le processus indique des erreurs ou avertissements. La distinction entre erreur standard et sortie standard permet de séparer facilement les deux types de messages.

En général, les *daemons* ne sont pas attachés à un terminal, et ils ne disposent donc pas de ces flux. Le cas des *daemons* invoqués par **inetd** est particulier : **inetd** les exécute en branchant leurs flux d'entrée/sortie sur une connexion réseau.

CULTURE **TCP et UDP**

TCP et UDP sont deux protocoles de transport de données fonctionnant avec IP, détaillés au chapitre 9. Sachez pour le moment que la machine dispose de 65536 ports TCP et d'autant de ports UDP, qui peuvent faire fonctionner des serveurs.

TCP et UDP fonctionnent sur IPv4 (l'actuelle version du protocole IP) ou IPv6 (sa nouvelle version, en cours de déploiement).

Plus fort que les *daemons* : le *super-daemon*

Le *super-daemon* est un *daemon* dont le travail est d'en invoquer d'autres. Appelé **inetd**, il a pour fichier de configuration /etc/inetd.conf.

Le *super-daemon* **inetd** est exécuté au démarrage sur pratiquement tous les systèmes Unix. Si tel n'est pas le cas, ou s'il a été tué, on peut le redémarrer en ligne de commande. En temps normal, il écoute les ports TCP et UDP indiqués dans son fichier de configuration. Lorsqu'une requête parvient sur l'un de ces ports, il invoque le cas échéant le *daemon* approprié pour la traiter.

L'intérêt du *super-daemon* est double : un seul processus écoute beaucoup de services, et simplifie ces derniers, puisque les *daemons* invoqués n'ont pas à se préoccuper du fonctionnement du réseau, mais se contentent de communiquer sur leurs entrées/sorties standard.

Le fichier de configuration se présente comme ceci :

```
ftp         stream  tcp     nowait  root    /usr/libexec/ftpd       ftpd -ll
#ftp        stream  tcp6    nowait  root    /usr/libexec/ftpd       ftpd -ll
#telnet     stream  tcp     nowait  root    /usr/libexec/telnetd    telnetd
#telnet     stream  tcp6    nowait  root    /usr/libexec/telnetd    telnetd
#shell      stream  tcp     nowait  root    /usr/libexec/rshd       rshd -L
#shell      stream  tcp6    nowait  root    /usr/libexec/rshd       rshd -L
```

Chaque ligne décrit un service ; les colonnes sont séparées par des blancs. On trouve, dans l'ordre :

- Le nom du service. La correspondance entre les noms et les ports est donnée par le fichier /etc/services. Sur certaines versions du super-serveur **inetd**, on peut préciser directement un numéro de port, mais sur d'autres, il faudra indiquer un nom préalablement défini dans /etc/services.
- Le type de trafic : flux connecté ou datagrammes. On indique respectivement `stream` pour du TCP et `dgram` pour de l'UDP.
- Le protocole de transport, dont le numéro est défini dans /etc/protocols. Ces protocoles se réduisent pratiquement à `tcp`, `udp`, et `tcp6` ou `udp6` pour IPv6.

PLUS LOIN **Écrire un daemon pour `inetd`**

Comme on le mentionnait précédemment, il est très simple avec **inetd** d'écrire un serveur : un script shell suffit ! Mettons en place un *daemon* répondant « *Hello world* » à toute connexion sur le port 5000.

On ajoute la ligne suivante dans /etc/services :

```
hello           5000/tcp
```

ainsi que la ligne suivante dans /etc/inetd.conf :

```
hello   stream   tcp   nowait   root   /tmp/hello.sh   hello.sh
```

/tmp/hello.sh est le script suivant :

```
#!/bin/sh
echo "Hello world"
```

Il ne reste plus qu'à signaler à **inetd** la modification pour tester ce nouveau serveur :

```
# ps -ax|grep inetd
 322 ?? Is    0:00.09 /usr/sbin/inetd -l
1466 p1 R+    0:00.13 grep inetd
# kill -1 322
# telnet localhost 5000
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Hello world
Connection closed by foreign host.
#
```

Dans le cas de programmes interactifs, on prendra garde aux problèmes de sécurité. Un utilisateur malicieux peut en effet essayer d'envoyer des données pour provoquer des fonctionnements non prévus du serveur.

Sur FreeBSD, on peut préciser `tcp46` ou `udp46` pour activer le service à la fois en IPv4 et en IPv6.

- Le mot-clef `nowait` ou `wait`, respectivement suivant qu'il faille ou non accepter de nouvelles connexions sur le port pendant le traitement d'une requête.
- L'identifiant sous lequel le processus gérant la requête doit être invoqué. **inetd** fonctionne sous l'UID de root pour pouvoir écouter sur tous les ports, y compris ceux inférieurs à 1024, ainsi que pour pouvoir modifier l'UID des *daemons* qu'il exécute. Mais ces derniers n'ont pas tous l'obligation de fonctionner sous l'UID de root.
- Le chemin de la commande à exécuter pour traiter la requête.
- Les arguments à donner à la commande.

Le fichier comprend de nombreuses entrées, toutes commentées par un caractère dièse (**#**) sur les systèmes récents. Dans l'exemple précédent, seul le service FTP en IPv4 est activé. FTP proprement dit se configure dans les fichiers de configuration du *daemon* **ftpd** ; le rôle d'**inetd** se limite à invoquer **ftpd** quand il reçoit des requêtes sur le port 21.

On active ou désactive un service en décommentant ou commentant son entrée dans le fichier /etc/inetd.conf avec un éditeur de texte, tel que **vi**. Pour que cette modification soit prise en compte, il faut envoyer à **inetd** le signal `SIGHUP` (de numéro 1), grâce à la commande **kill** :

```
# ps -ax | grep inetd
 322 ?? Is   0:00.09 /usr/sbin/inetd -l
1454 p1 S+   0:00.14 grep inetd
# kill -1 322
```

Sur certains systèmes GNU/Linux récents, **inetd** est remplacé par **xinetd**. Ce dernier joue le même rôle, mais, dans une optique de souplesse, il utilise un fichier de configuration par service. Ces fichiers sont placés dans /etc/xinetd.d. **xinetd** apporte aussi quelques nouvelles fonctionnalités, dont la possibilité de n'écouter que sur certaines adresses (comme 127.0.0.1). Son gros inconvénient est son format de configuration, différent de celui d'**inetd** : les ingénieurs système ont souvent horreur de manipuler des programmes différents pour remplir la même fonction.

PLUS LOIN **inetd et xinetd**

Le super-serveur **inetd** des systèmes BSD a peu à envier à **xinetd** : on peut par exemple lui indiquer de ne fournir un service que sur une adresse IP donnée. Exemple de ligne inetd.conf pour obtenir un *daemon* FTP n'écoutant que sur l'interface d'adresse `10.0.0.1` :

```
10.0.0.1:ftp   stream  tcp     nowait  root    /usr/libexec/ftpd     ftpd -ll
```

Il s'agit d'une extension BSD, absente des **inetd** d'autres systèmes.

PLUS LOIN **TCP wrappers**

On trouve parfois en sixième colonne, précisant la commande à invoquer, **tcpd** (du paquetage *TCP wrappers*). Elle permet d'ajouter des contrôles sur les connexions via les fichiers /etc/hosts.allow et /etc/hosts.deny, pour par exemple constituer une ACL (liste de contrôle d'accès) ou interdire les connexions en provenance de machines non résolues dans le DNS.
Sur les systèmes BSD, ces fonctionnalités étant intégrées dans **inetd**, on peut en bénéficier sans faire appel à **tcpd**.

SÉCURITÉ **Les ports inférieurs à 1024**

Les ports TCP et UDP inférieurs à 1024 sont réservés à root : seuls les processus employant son UID ont le droit de les utiliser.

RAPPEL **Port 21 ?**

Le port 21 est le port par défaut pour FTP, tel que l'indique le fichier /etc/services.

ATTENTION **La commande `kill` peut tuer**

kill permet d'envoyer des signaux à un processus pour lui notifier différentes situations, telles qu'un changement de configuration, ou lui enjoindre de finir son exécution. Il existe donc plusieurs signaux, que l'on peut indiquer en option : **-1** (`SIGHUP`) est souvent utilisé pour indiquer aux *daemons* de relire leur configuration. **-15** (`SIGTERM` ; signal utilisé par défaut si on ne précise pas de numéro de signal à **kill**) indique à un processus que l'on souhaiterait qu'il se termine au plus vite. **-9** (`SIGKILL`) tue le processus immédiatement et sans appel.
On réservera **kill -9** aux processus irrémédiablement plantés, sourds à un **kill** normal. Une utilisation systématique de **kill -9** pourrait causer des dégâts sur les données manipulées par certains programmes.
En bref, n'oubliez pas de préciser le signal **-1**, sans quoi vous tuerez **inetd** !

Et dites à M. Cron de cesser de m'écrire tous les jours !

Autre *daemon* exécuté par défaut sur pratiquement tous les systèmes Unix : **cron**. Il a pour rôle d'exécuter des tâches à intervalles de temps réguliers, et trouve sa configuration dans des fichiers appelés crontab. Chaque utilisateur peut en avoir un, pour peu qu'on lui donne accès aux commandes adéquates.

Sous BSD, les fichiers crontab sont placés dans /var/cron/tabs. Ils ont pour allure :

```
SHELL=/bin/sh
PATH=/bin:/sbin:/usr/bin:/usr/sbin
HOME=/var/log
#
#minute hour    mday    month   wday    command
#
0       *       *       *       *       /usr/bin/newsyslog
```

On trouve deux types de lignes (hormis les commentaires) : des définitions de variables d'environnement et des lignes de configuration de commandes à exécuter régulièrement. Ces dernières comportent 6 colonnes, séparées par des blancs. Les 5 premiers champs indiquent les moments auxquels la commande indiquée dans le sixième champ doit être invoquée, un astérisque (*****) indiquant toutes les valeurs possibles :

- la minute (de 0 à 59)
- l'heure (de 0 à 23)
- le jour du mois (de 1 à 31)
- le mois dans l'année (de 1 à 12)
- le jour de la semaine (de 0 à 7 ; 0 et 7 sont dimanche, 1 est lundi, etc.)

Toutes les erreurs et sorties standard des programmes exécutés ainsi sont envoyées par courrier électronique au propriétaire de la crontab. Pour éviter de recevoir de tels messages tous les jours, voire toutes les heures ou toutes les minutes, on redirigera la sortie des programmes dans un fichier, ou /dev/null pour ne plus en entendre parler (avec le risque de mettre plus longtemps à découvrir un possible dysfonctionnement) :

```
0     *     *     *     *     /home/manu/truc.sh 2>&1 > /home/manu/truc.log
```

On ne modifie pas directement les crontab : il faut passer par la commande **crontab**, qui permet de lister le contenu d'une crontab (**crontab -l**) ou de l'éditer (**crontab -e**) avec son éditeur favori. Après cette session d'édition, **crontab** contrôle la syntaxe et prévient **cron** de la modification. Le super-utilisateur peut comme souvent intervenir en lieu et place des utilisateurs, grâce à l'option **-u**.

Dans les systèmes BSD, seul root a par défaut un fichier crontab activé, utilisé pour faire fonctionner les tâches en attente pour **at** (consulter à ce sujet sa page **man**), les journaux avec **newsyslog**, et les vérifications quotidienne, hebdomadaire et mensuelle du système (un coup d'œil dans la boîte aux lettres de root vous instruira rapidement à leur sujet, un rapport étant émis pour chacune d'entre elles).

EN CAS DE COUP DUR **À quelle heure êtes-vous ?**

Si les crontab ne s'exécutent pas au moment souhaité, on se posera les questions suivantes : l'horloge est-elle à l'heure ? Le fuseau horaire est-il bien configuré ? Ce réglage, propre à chaque utilisateur, se fait via la variable d'environnement TZ (*Time Zone*). Exemple : **TZ=Europe/Paris**

TZ indique le chemin d'accès à un fichier de description d'heure locale placé dans /usr/share/zoneinfo. La configuration ci-dessus choisit le fichier /usr/share/zoneinfo/Europe/Paris, qui donne l'heure de France métropolitaine, avec la gestion des heures d'été et d'hiver.

Lorsque TZ n'est pas définie, c'est le lien /etc/localtime qui indique le fuseau horaire par défaut.

SUR LES AUTRES UNIX **cron**

Certains Unix disposent d'un **cron** plus fruste, qui ne permet pas de définir des variables d'environnement dans les fichiers crontab.

PLUS LOIN **Shell et redirection d'erreur standard**

La notation **2>&1** permet de rediriger le flux d'erreur standard dans le flux de sortie standard, en les fusionnant tous deux dans ce dernier. Cette écriture n'est valable qu'en **sh**, **ksh** et **bash** : **cron** utilise /bin/sh pour exécuter les commandes.

/dev/null est un fichier spécial qui se comporte comme un trou noir : tout ce qui y est écrit est ignoré. Il est très utile pour faire taire les programmes trop loquaces.

Consignation des événements avec `syslogd`

Sous Unix, les programmes sont très bavards, et consignent de nombreux événements : heures et terminaux de connexion des utilisateurs, passages de courrier électronique dans la messagerie, messages d'alerte ou d'erreur des services...

Ces informations sont enregistrées dans un journal, ou envoyées à **`syslogd`**, *daemon* chargé de centraliser les journaux. Il enregistre ces informations dans des fichiers, conformément à sa configuration, précisée dans le fichier /etc/syslog.conf.

`syslogd` est un *daemon* autonome, qu'on peut exécuter en ligne de commande. On l'invoque la plupart du temps avec l'option **`-s`**, qui l'empêche d'accepter les informations de journalisation en provenance d'autres machines sur le réseau. Pour lui faire relire son fichier de configuration, il faut employer **`kill -1`**, comme pour **`inetd`** et **`init`**.

Extrait de fichier syslog.conf ;

```
*.info                  /var/log/messages
mail.info               /var/log/maillog
*.notice                root
*.*                     /dev/ttyE6
```

On trouve deux champs :

- Catégorie et niveau d'importance, séparés par un point. On inscrira un astérisque pour concerner toutes les valeurs possibles. Demander un niveau d'importance donné impliquera aussi tous les niveaux supérieurs.
- Destination. Il peut s'agir d'un fichier, d'un terminal, ou d'un identifiant de connexion, auquel cas l'utilisateur verra passer ces informations sur son terminal lorsqu'il sera connecté.

La liste des catégories et des niveaux se trouve dans la page **`man`** de syslog.conf.

SUR LES AUTRES UNIX **syslog.conf et syslogd.conf**

Sur certains Unix, le fichier de configuration de **`syslogd`** s'appelle syslogd.conf, et non syslog.conf.

RAPPEL **`init`**

Le processus **`init`** a été largement abordé au chapitre 5. Sur les BSD, son rôle se limite à invoquer le script d'initialisation /etc/rc et à gérer les invites de connexion sur les terminaux de la machine.

PLUS LOIN **Faire tourner les journaux**

Quand les journaux se remplissent, ils occupent de l'espace disque. Travailler sur un fichier énorme est pénible, et il est pratique de compresser les anciens journaux, peu consultés. Pour éviter de remplir le disque, il faudra de plus effacer les journaux très anciens.
Les programmes **`logrotate`** ou **`newsyslog`** assurent ces tâches. Sur les BSD, **`newsyslog`** est intégré au système, et invoqué à intervalles de temps réguliers par le fichier crontab de root pour faire le ménage. **`newsyslog`** se configure dans le fichier /etc/newsyslog.conf, dont on consultera la page **`man`** pour modifier la configuration.

Appels de procédures distantes

Les RPC (*Remote Procedure Call*, ou appels de procédures distantes) sont un mécanisme de communication inter-processus capable de fonctionner à travers le réseau. Ils permettent à un processus de faire un appel de fonction qui s'exécute sur une autre machine.

Les serveurs utilisant les RPC font appel à des ports TCP et UDP alloués dynamiquement, et un *daemon* assure un service d'annuaire pour que les clients puissent les contacter. Deux *daemons* sont capables de remplir cette fonction : **`portmap`**, et **`rpcbind`**.

`rpcbind` doit être invoqué avant tout serveur utilisant les RPC. Si le système ne l'exécute pas dans la configuration par défaut, il est possible d'y remédier en l'invoquant à la ligne de commande. Pour le configurer au démarrage sur NetBSD, on ajoutera la ligne **`rpcbind=YES`** au fichier /etc/rc.conf. Pour FreeBSD, il faut écrire **`rpcbind_enable=YES`**.

EN PRATIQUE **`portmap` et `rpcbind`**

Ces deux *daemons* ont exactement le même rôle, **`rpcbind`** est simplement plus récent ; on le préférera donc si on a le choix. À l'heure où sont écrites ces lignes, NetBSD et FreeBSD utilisent **`rpcbind`** et OpenBSD utilise toujours **`portmap`**.
Nous utiliserons dans la suite de ce livre l'exemple de **`rpcbind`**, mais on pourra lui substituer systématiquement **`portmap`**.

ALTERNATIVE **D'autres systèmes de RPC**

Il existe plusieurs types de RPC. Sous Unix, c'est *Sun Microsystems* qui est à l'origine de l'implémentation la plus utilisée, parfois appelée SunRPC.
L'*Open Software Foundation* (OSF) est à l'origine d'une autre implémentation des RPC, incompatible avec celle de Sun : les RPC DCE. Leur plus gros utilisateur est sans nul doute Windows, qui s'en sert comme base pour tout son environnement de communication interprocessus distribué DCOM.
Les RPC sont taillés pour la programmation procédurale. En programmation objet, ils ont pour équivalent les invocations de méthodes distantes (RMI, ou *Remote Method Invocations*) de Java, que l'on doit encore à *Sun Microsystems*. L'environnement de communication inter-processus distribué CORBA propose lui aussi son mécanisme d'invocation de méthodes distantes, indépendant du langage et de la plateforme hébergeant les processus.

Sur les UNIX System V et GNU/Linux, **rpcbind** s'active ou se désactive en plaçant ou en détruisant un lien symbolique dans l'un des répertoires /etc/rcX.d.

Les clients des RPC n'ont pour leur part besoin d'aucun *daemon* pour fonctionner : ils se contentent d'interroger le **rpcbind** de la machine cible pour contacter le serveur RPC.

rpcbind ne fournit aucun service directement utile à l'utilisateur, mais il est indispensable à des services comme le partage de fichiers en NFS.

Réseaux de machines Unix

Dans certaines situations, il faut banaliser les machines d'un réseau : en se connectant sur n'importe quel poste, l'utilisateur doit retrouver son répertoire et ses fichiers personnels. On peut aussi vouloir répartir sur un ensemble de machines les informations de *login*, les groupes... ou tout à la fois. Cette section traite de ces problèmes, tant du côté du client que du côté serveur.

Synchronisation des horloges

Dans le cas d'une machine isolée, il est utile d'être à l'heure. Dans le cas de machines partageant des systèmes de fichiers, cela devient critique, pour éviter toute une kyrielle de comportements étranges. Les mécanismes d'authentification forte (SSH, Kerberos) exigent souvent, eux aussi, une synchronisation des horloges.

On peut procéder de différentes façons. La commande **rdate**, par exemple, interroge un serveur de temps assurant le service `time` (activable via **inetd**). On peut exécuter **rdate** régulièrement via un fichier crontab pour maintenir les pendules à l'heure.

EXEMPLE **Dérives d'horloge**

Sous une forte charge, un Macintosh II sous NetBSD ou OpenBSD peut dériver de plusieurs secondes par jour. Pour ces machines, une synchronisation toutes les 24 heures est inadéquate.

Se contenter de **rdate** pose quelques problèmes. Si on ne l'invoque qu'une fois par jour, on pourra ressentir des dérives dues à des horloges système médiocres. De plus, **rdate** met l'horloge à l'heure de manière brutale, sans hésiter à lui faire remonter le temps le cas échéant. Certaines applications utilisant le couple (PID du processus, heure système) comme identifiant unique sur le système ont absolument horreur de cela.

Une meilleure solution consiste à utiliser un *daemon* de synchronisation des horloges, comme **timed** ou **ntpd**. Ils accélèrent ou ralentissent l'horloge système pour rattraper le temps donné par un serveur de temps. La synchronisation étant continue, on n'observe jamais de saut brusque.

CULTURE **Horloge atomique**

Certains phénomènes physiques se déroulent avec une fréquence propre très précise ; on peut donc les utiliser pour construire des horloges. La vibration du cristal de quartz est très utilisée pour construire des horloges de qualité moyenne : en 6 ans de fonctionnement, elles dérivent d'une seconde au maximum.
Les horloges atomiques utilisent la fréquence de transition d'un niveau d'énergie de l'atome de césium. Le précision est de l'ordre d'une seconde de dérive pour 3000 ans de fonctionnement.

timed implémente un protocole de synchronisation d'horloge un peu tombé en désuétude : TSP. **ntpd** implémente NTP (*Network Time Protocol*), largement utilisé sur Internet : on trouve des serveurs NTP reliés à des horloges atomiques, ou en recevant le signal par radio. Ces serveurs sont dits de strate 1, et servent de serveur de temps à des serveurs de strate 2, eux-mêmes serveurs de temps de serveurs de strate 3, etc. Le système est organisé de façon hiérarchique pour éviter que les serveurs de strate 1 n'aient à donner l'heure à toutes les machines sur Internet.

On peut employer NTP en circuit fermé, pour synchroniser les horloges des machines d'un site. L'heure obtenue risque de flotter légèrement par rapport à l'heure réelle, mais les horloges des machines seront à la même heure. On peut aussi indiquer aux serveurs **ntpd** de se synchroniser sur un serveur de temps sur Internet, pour rester à l'heure réelle. La liste des serveurs publics est disponible à l'adresse http://www.ntp.org

ntpd s'invoque en ligne de commande, ou au démarrage des BSD en indiquant **ntpd=YES** dans /etc/rc.conf (**xntpd_enable="YES"** pour FreeBSD). Il lit sa configuration dans le fichier /etc/ntp.conf. Les possibilités sont nombreuses : on peut par exemple utiliser des clefs RSA pour authentifier le serveur de temps, évitant ainsi les attaques fondées sur les perturbations de l'horloge système.

Pour provoquer le lancement de **ntpdate** au démarrage, il suffit de modifier le /etc/rc.conf. Pour NetBSD et FreeBSD, **ntpdate** utilisera le serveur NTP indiqué dans /etc/ntpd.conf, il suffit d'indiquer **ntpdate=YES** pour NetBSD, et **ntpdate_enable="YES"** pour FreeBSD. Sur OpenBSD, on indique dans le /etc/rc.conf le serveur NTP que **ntpdate** doit contacter, par exemple **ntpdate_flags=ntp.euro.apple.com**.

EN PRATIQUE **Serveur de temps**

Voici quelques serveurs de temps public de strate 2, à toutes fins utiles :

- ntp1.belbone.be
- ntp.euro.apple.com
- ntp.univ-lyon1.fr
- ntp2a.mcc.ac.uk

PIÈGE À C... **Invocation de ntpd**

ntpd refusera de fonctionner si l'heure locale est trop éloignée de l'heure du serveur de temps. Pour régler ce problème, on utilise souvent la commande **ntpdate** au démarrage, qui remet brutalement à l'heure l'horloge système en utilisant un serveur NTP. Utilisée avant le démarrage de services sensibles aux variations temporelles brusques, elle ne causera pas de problèmes.

Synchronisation des comptes, groupes, et autres

Les informations des comptes, les groupes, les tables d'alias de messagerie, ... peuvent être répartis entre les machines. Là encore, les manières de procéder sont nombreuses. On peut par exemple copier les fichiers concernés via SSH, avec la commande **scp**. À l'aide de clefs RSA, on peut rendre le processus totalement automatique et ne jamais avoir à fournir un mot de passe. Seul souci : cela manque de souplesse. Pour conserver des comptes locaux, comme pour root, il faudra bricoler les fichiers de mots de passe avec des scripts, manipulation dangereuse : si elle tourne mal, on risque de ne plus pouvoir se connecter sur la machine.

L'alternative est d'utiliser un système de répartition bien rôdé. Ils sont nombreux : Kerberos, LDAP, NIS... À chacun son champ d'application, mais NIS reste le plus simple à mettre en œuvre. C'est de plus la méthode historique, la plus répandue sur les systèmes Unix ; c'est celle que nous détaillerons.

NIS permet de répartir de manière assez simple bien des éléments entre les machines : comptes, groupes, bases d'alias de messagerie, noms de machines... Son seul défaut est de tout envoyer en clair à travers le réseau, et notamment la base de mots de passe chiffrés – ce qui le rend peu sûr. Pour une meilleure sécurité, utilisez NIS avec IPSec (protocole de réseau sécurisé dont nous parlerons au chapitre 9), ou tournez-vous vers d'autres solutions, telles que LDAP.

NIS diffuse les informations présentes dans un certain nombre de fichiers appelés tables NIS (*NIS maps* en anglais). Elles sont réparties sur toutes les machines du domaine NIS. Ce domaine a un nom, que toutes les machines doivent partager, et qu'on peut configurer à la main avec la commande **domainname**. La reconfiguration automatique au démarrage peut se faire via /etc/rc.conf sous BSD : il faut renseigner la variable `domainname` sur NetBSD, ou `nisdomainname`

CULTURE **De Yellow Pages à NIS**

NIS (*Network Information Service*, ou service d'informations de réseau) fut introduit par *Sun Microsystems* dans SunOS. À l'époque, NIS s'appelait *Yellow Pages* (pages jaunes), mais Sun a dû changer de nom car c'était une marque déposée de *British Telecom*. Le nom actuel est NIS, et l'ancienne dénomination explique que toutes les commandes liées à NIS soient préfixées par les deux lettres **yp**.

SÉCURITÉ **Mots de passe chiffrés en clair ?**

On peut se demander où est le mal lorsque des mots de passe chiffrés sont transmis en clair sur le réseau. Après tout, ils sont déjà chiffrés : que faire de plus ?

Le problème est que le chiffrement standard des mots de passe par l'algorithme DES n'est pas très robuste. Un espion motivé pourra casser relativement rapidement des mots de passe. Pour des raisons d'interopérabilité, il est souvent difficile d'utiliser un autre algorithme de chiffrement en combinaison avec NIS.

sur FreeBSD). Il n'est pas obligatoire de choisir un domaine NIS identique au domaine DNS.

Un domaine NIS s'articule en trois niveaux :

- Le serveur NIS maître, qui emploie un serveur **ypserv**, et sur lequel est construite toute version maître des tables NIS.
- Un nombre positif ou nul de serveurs NIS secondaires, dont le rôle est de remplacer le serveur NIS maître en cas de panne. Ces serveurs emploient également le *daemon* **ypserv**. Lorsque le serveur maître doit mettre à jour ses tables NIS, il pousse les nouvelles tables vers ses serveurs esclaves avec la commande **yppush**.
- Les clients NIS, qui emploient le *daemon* **ypbind**. Lorsqu'un processus a besoin d'une information répartie par NIS, comme un mot de passe ou un nom de groupe associé à un GID, il interroge le *daemon* **ypbind**, qui interroge à son tour un serveur NIS.

  À tout moment il faut donc qu'un serveur NIS soit disponible, sans quoi les clients resteront bloqués sur leurs requêtes.

ATTENTION **ypserv et ypbind utilisent les RPC**
NIS fonctionne avec des appels de procédures distantes. Comme tous les serveurs utilisant des RPC, **ypserv** et **ypbind** ont besoin de **portmap** ou **rpcbind** pour fonctionner correctement.

La mise en place de NIS ne se limite pas à invoquer un serveur **ypserv** ou **ypbind**. Il faut aussi mettre en forme les tables NIS et les répertoires qui les accueillent, dans /var/yp. La commande **ypinit** est utilisée pour cela. Suivant les options avec lesquelles elle est invoquée, elle mettra en place le système pour qu'il se comporte en client, en serveur maître, ou en serveur esclave.

Dernière étape, il faut indiquer quelles informations seront traitées par NIS. Historiquement, la répartition des comptes d'utilisateurs et des groupes se faisait en ajoutant à la fin du fichier concerné une entrée dont tous les champs étaient vides, à l'exception du premier, de valeur **+**. Exemple pour le fichier /etc/passwd :

```
(...)
manu:pCs71RItmHgdc:500:500:Emmanuel Dreyfus:/home/manu:/bin/ksh
+::::::
```

SÉCURITÉ **Table passwd**
NIS est peu sûr. On peut le sécuriser en lui adjoignant IPSec, mais il a encore d'autres défauts. Par exemple, taper **ypcat passwd** sur un client NIS sortira la table passwd, fichier /etc/passwd ancien style contenant les mots de passe chiffrés. C'est gênant.
Les systèmes BSD proposent une réponse commune à ce problème : une table passwd sans mot de passe, lisible par tous, et une table master.passwd avec mots de passe, lisible uniquement par root (mais qui traversera le réseau en clair). On peut activer ce comportement en retouchant /var/yp/Makefile.yp et en inscrivant **INSECURE=no**, réglage qui ne fonctionne qu'avec des clients et des serveurs BSD.

Cette syntaxe est encore reconnue sur les systèmes BSD, mais il existe aussi un mécanisme plus général : le fichier /etc/nsswitch.conf. Il permet d'indiquer quelles informations doivent être prises dans des fichiers, via NIS, ou via le DNS. Sa page **man** fournira davantage d'informations.

Répartition du système de fichiers

Évoquons pour conclure la répartition du système de fichiers. Le but est de monter une partie de l'arborescence d'une machine sur les autres, comme par exemple le /home.

Ici encore, plusieurs choix sont disponibles : NFS (*Network FileSystem*, ou système de fichiers en réseau), AFS (*Andrew FileSystem*, ou système de fichiers d'Andrew), Coda... NFS étant le système historique le plus fréquemment rencontré, c'est celui que traiterons.

Le service NFS lui-même est assuré par le *daemon* **nfsd**. Il se fait généralement en UDP, voire en TCP pour les systèmes récents. TCP permet d'obtenir des montages NFS de bonne qualité à travers Internet, et UDP est plus économe en ressources.

Sur les BSD, **nfsd** démarre par défaut avec les options **–tun4**. Sa page **man** vous en exposera la signification.

Pour manipuler un fichier d'une arborescence montée par NFS, le client dialogue avec le *daemon* **nfsd** du serveur.

Mais le montage de l'arborescence NFS elle-même se fait par un autre protocole, pour lequel le serveur est le *daemon* **mountd**. Sa configuration est stockée dans le fichier /etc/exports. Ce dernier précise quelles arborescences sont exportées par NFS, et avec quelles options. Lors de toute modification, il faut envoyer le signal **kill –1** à **mountd** pour qu'elle soit prise en compte.

Le format de ce fichier est assez simple : des répertoires, éventuellement des options, suivies le cas échéant de la liste des machines concernées. Voyez la page **man** exports pour les détails des options disponibles.

```
/home –network 192.168.15.0 –mask 255.255.255.0
/usr –ro –mapall=nobody
```

Pour vérifier la configuration /etc/exports, on peut utiliser la commande **showmount –e** : elle permet d'interroger un *daemon* **mountd** à travers le réseau pour savoir quels répertoires sont exportés. Exemple :

```
$ showmount –e localhost
Exports list on localhost:
/cdrom                          10.0.12.48 10.0.12.2
$ showmount –e 10.0.12.2
Exports list on 10.0.12.2:
/pub/tftp                       Everyone
/pub/nfs/indy                   10.0.12.48
```

IMPORTANT **nfsd, mountd, et les RPC**

Comme **ypserv** et **ypbind**, **nfsd** et **mountd** utilisent les appels de procédures distantes. Il faudra donc invoquer **portmap** ou **rpcbind** avant de pouvoir les utiliser.

SÉCURITÉ **Montages NFS**

Comme NIS, NFS est peu sûr. Son principal défaut : c'est le client qui traite l'authentification. S'il prétend que tel utilisateur a l'UID 501, le serveur le croira, que cela soit vrai ou pas. Un utilisateur malicieux disposant d'un accès root sur une machine du réseau pourra donc simuler n'importe quel identifiant utilisateur auprès du serveur NFS.

Pour limiter les dégâts, NFS donne par défaut les droits de l'utilisateur *nobody* (en anglais, « personne », UID 32767) à tout utilisateur prétendant avoir un UID nul. De ce fait, root n'aura pas les droits de root sur un répertoire monté par NFS, ce qui empêchera d'y déposer un shell *set-UID* root. Si on laissait cette possibilité, il serait possible de se connecter sur le serveur NFS avec un compte non privilégié et de passer root en exécutant ce shell.

Malgré cela, un utilisateur qui serait root sur un client et disposant d'un compte sur le serveur pourrait encore devenir l'utilisateur *daemon* ou *operator* (opérateur), ce qui n'est pas bon pour la sécurité du système : *operator* peut lire les disques directement, via les fichiers spéciaux du /dev ; il peut donc en particulier rapatrier le fichier des mots de passe.

Une mesure de sécurité consiste donc à interdire systématiquement l'exécution des fichiers situés sur un répertoire exporté par NFS. On procède avec l'option de montage **noexec** dans le fichier /etc/fstab du serveur NFS. Si cela n'est pas possible, les options **nosuid** et **nodev** peuvent êtres bonnes à prendre.

Bien sûr, les choses sont considérablement simplifiées dès lors qu'aucun utilisateur ne peut ouvrir de session sur le serveur NFS lui-même, ou si on n'utilise que des exports NFS en lecture seule. Mais d'autres problèmes de sécurité demeurent, comme le fait que tout le trafic passe en clair.

B.A.-BA **Verrous**

NFS en lui-même ne prévoit pas de mécanisme de verrouillage pour empêcher deux processus d'écrire à la fois dans le même fichier. Ce genre d'opération concurrente, rarement souhaitable, mène souvent à la corruption du fichier.
Pour éviter cela, on pose des verrous à l'aide du *daemon* **rpc.lockd**.

PERFORMANCES **nfsiod sur le client**

Le *daemon* facultatif **nfsiod** permet d'améliorer les performances des clients NFS. Il s'exécute en ligne de commande, sans options particulières, ou via les scripts de démarrage.
Sur certains systèmes, **nfsiod** est un processus intégré au noyau, qui démarre tout seul lorsque le besoin s'en fait sentir.

PIÈGE À C... **Montages NFS soft**

L'option de montage **-s** sur NetBSD, ou **soft** sur FreeBSD, permet d'obtenir un montage « *soft* » (doux), qui échouera si le serveur est indisponible. Elle évite de rester bloqué sur les requêtes NFS en cas de problème réseau.
Ce comportement peut paraître désirable, mais il a des effets secondaires : la plupart des programmes Unix n'ont pas été programmés dans la perspective de voir leurs opérations d'entrées/sorties échouer. Certains programmes gèrent très mal un échec sur une opération d'écriture, et peuvent alors laisser des fichiers corrompus ou tronqués.
Pour ne pas rester bloqué en cas d'indisponibilité du serveur, utilisez plutôt des montages interruptibles.

Enfin, pour permettre aux clients de poser des verrous sur les fichiers montés en NFS, il faut employer les *daemons* **rpc.lockd** et **rpc.statd** sur le serveur (on les exécute en ligne de commande ou au démarrage, avec /etc/rc.conf).

Sur le client, en comparaison, les choses sont très simples. Aucun *daemon* n'est obligatoire, Il suffit de monter le répertoire, ce qu'on peut faire à la main, comme suit : **mount -t nfs crapulax:/home /home**

Pour obtenir le montage automatique au démarrage, il faut l'indiquer dans /etc/fstab. Voici un exemple, avec des options de montage valables pour NetBSD :

```
/dev/sd0a       /       ffs     rw 1 1
/dev/sd0b       none    swap    sw 0 0
crapulax:/home  /home   nfs     rw,-T,-i,-b 0 0
```

Sous FreeBSD, la même configuration s'obtient avec le /etc/fstab suivant :

```
/dev/da0s1a      /       ffs     rw 1 1
/dev/da0s1b      none    swap    sw 0 0
crapulax:/home   /home   nfs rw,tcp,intr,bg 0 0
```

Dans les exemples ci-dessus, on a indiqué quatre options à l'intention de **mount_nfs**, invoqué au démarrage pour procéder au montage :

- **-T** sous NetBSD, ou **tcp** sous FreeBSD : demande un montage NFS via TCP.
- **-i** sous NetBSD, ou **intr** sous FreeBSD : rend les opérations NFS interruptibles : un processus bloqué sur une entrée/sortie NFS pourra être interrompu par le signal SIGTERM (combinaison de touches **Control**+**C**). En son absence, il faut attendre la fin de l'opération pour arrêter le processus, et si un serveur est rendu indisponible, le processus concerné risque de rester bloqué longtemps.
- **-b** sous NetBSD, ou **bg** sous FreeBSD : demande un montage en arrière-plan : si au démarrage le serveur n'est pas disponible, les choses suivent leur cours, et **mount_nfs** est exécuté en tâche de fond pour procéder au montage NFS une fois que le serveur sera disponible. Dans une situation où des machines Unix font des montages NFS réciproques, cette option est plus que recommandée, car c'est le seul moyen de redémarrer toutes les machines après une coupure de courant en évitant un interblocage.

Le principe est le même sur les autres systèmes, mais on en consultera les pages **man** pour connaître les noms des options équivalentes.

PIÈGE À C... **Blocages dus à NFS**

NFS incite à monter des réseaux complexes, avec des situations d'interdépendances (quand plusieurs machines ont besoin les unes des autres pour démarrer). Après une coupure de courant, toutes les machines démarrent en même temps, et la situation est bloquée. On résout ces situations avec des montages interruptibles, et en plaçant le montage en tâche de fond au démarrage.

Malheureusement, si le DNS n'est pas disponible, les noms de machines indiqués dans le /etc/fstab provoqueront des requêtes DNS qui, faute de réponse, bloqueront le démarrage. La solution consiste soit à utiliser des adresses IP dans /etc/fstab, soit à indiquer les adresses IP des machines indispensables au démarrage dans /etc/hosts.

Netboot

L'intégration des machines Unix entre elles va parfois jusqu'à l'extrême : le *netboot* (démarrage par réseau). Voyons rapidement les services utiles au *netboot* et à la configuration d'autres machines sur le réseau.

Les *daemons* **dhcpd** et **bootpd** permettent de configurer automatiquement des machines sur le réseau. Le principe est relativement simple : la machine en demande de configuration envoie à tout le réseau (*broadcast*) une requête DHCP, que le serveur intercepte. Il renvoie ensuite les éléments de configuration de réseau à la machine émettrice. **dhcpd** est un *daemon* à invoquer au démarrage (ou à la ligne de commande), et qui lit sa configuration dans /etc/dhcpd.conf. Pour apprendre à configurer **dhcpd**, reportez-vous à la page **man** de dhcpd.conf. Attention, **dhcpd** n'a pas prévu de recharger sa configuration lorsqu'il reçoit un SIGHUP. Pour notifier **dhcpd** d'un changement de configuration, il faut obligatoirement le tuer et l'exécuter à nouveau.

BootP est l'ancêtre de DHCP. Le *daemon* **bootpd** est invoqué via **inetd**; il suffit donc de l'activer dans /etc/inetd.conf. **bootpd** lit sa configuration dans /etc/bootptab, fichier dont on consultera la page **man** pour apprendre à le configurer. **bootpd** étant exécuté à chaque requête, il n'y a rien à faire pour que les modifications apportées à /etc/bootptab soient prises en compte.

Certaines machines démarrent avec un protocole encore plus ancien : RARP (*Reverse Address Resolution Protocol*, ou protocole de résolution inverse des adresses), dont les requêtes sont gérées par le *daemon* **rarpd**. Comme **dhcpd**, c'est un *daemon* autonome (indépendant du super-serveur **inetd**). Sa configuration se trouve dans /etc/ethers.

Après configuration de leurs interfaces, les machines faisant appel au *netboot* peuvent avoir besoin de télécharger un noyau ou des fichiers de configuration via TFTP. C'est le *daemon* **tftpd** qui assure ce service. Invoqué par **inetd**, Il n'a aucun fichier de configuration, et prend toute sa configuration sur la ligne de commande (elle consiste essentiellement en le répertoire où se trouvent les fichiers à servir).

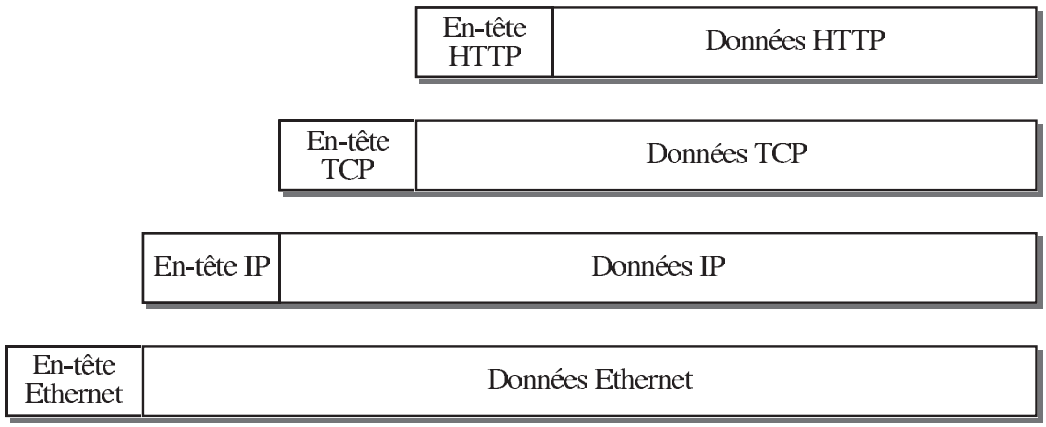
Enfin, le *netboot* d'une machine Unix passera par le montage à travers le réseau de la racine, voire de la mémoire de pagination (*swap* en anglais). Ceci se fait via le protocole NFS, dont nous avons précédemment vu la configuration sur un serveur : il faut remplir le fichier /etc/exports et invoquer au moins les *daemons* **rpcbind** (ou **portmap**), **nfsd**, et **mountd**.

PLUS LOIN **Compléments de RARP**

RARP ne met en place que l'adresse IP ; le reste de la configuration sera découvert par un autre protocole. Certaines machines utilisent pour cela des appels de procédures distantes vers une machine du même réseau.
Le serveur répondant aux requêtes de *netboot* en RPC est **rpc.bootparamd**, de fichier de configuration /etc/bootparams. Comme tous les serveurs utilisant les RPC, **rpc.bootparamd** a besoin du *daemon* **portmap** ou **rpcbind** pour fonctionner.

9

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html><head><meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">



Le réseau pour les administrateurs pressés

Les systèmes Unix sont souvent intimement liés aux réseaux. Pour comprendre l'administration des clients et services réseau, la connaissance des protocoles impliqués est indispensable. Ce chapitre traite donc des différents protocoles réseau utilisés dans la vie quotidienne de l'administrateur système.

SOMMAIRE

MOTS-CLEFS

- Réseaux Ethernet
- Internet
- Protocoles applicatifs

Démêler les protocoles

Le monde du réseau regorge de protocoles différents, souvent désignés par des acronymes. Quelques noms sembleront peut-être familiers : TCP, IP, UDP, CSMA/CD, ARP, HTTP, DNS, RPC, DHCP, ICMP, RTSP... La liste pourrait s'allonger presque indéfiniment. La première difficulté pour le novice est donc de repérer le rôle de chacun de ces protocoles, pour situer rapidement tout nouveau protocole dans l'édifice.

Autre objectif : connaître le minimum vital sur les protocoles que l'on sera amené à utiliser. Le présent chapitre s'attache à remplir ces deux fonctions, malgré les contraintes de place. On se limitera donc à une introduction rapide mais utile sur les réseaux Ethernet et IP.

NE LISEZ PAS CE CHAPITRE !

Si vous aimez essayer de diagnostiquer des problèmes qui vous dépassent, si vous appréciez le plaisir subtil de ne rien comprendre à la documentation, ou de passer pour un inculte lors des discussions entre collègues, passez directement au chapitre suivant.

Bien entendu, le lecteur déjà au fait des bases des réseaux pourra faire l'impasse sur le présent chapitre sans s'exposer à ces déconvenues.

BIBLIOGRAPHIE **Pour en savoir plus**

Les curieux, avides de détails et de technologies exotiques, consulteront avec profit des ouvrages tels que le *Computer Networks* d'Andrew S. Tanenbaum.

Des piles de protocoles

Tous les protocoles s'utilisent les uns les autres : HTTP, qui transfère des pages Web, fait appel à TCP, garantissant un canal de communication fiable. Ce dernier fait appel à IP pour communiquer avec des machines partout sur Internet. Suivant la nature du réseau sous-jacent, IP reposera sur différents protocoles, tels que PPP pour une liaison téléphonique, ou HDLC sur une liaison spécialisée.

On voit se dessiner des empilements de protocoles, comme dans le cas présent HTTP/TCP/IP. C'est toujours ainsi que fonctionnent les réseaux : un protocole assure un service, et s'appuie sur les services fournis par un autre protocole.

Le monde IP comprend plusieurs familles de protocoles. Les protocoles applicatifs assurent un service utilisé directement par les applications : HTTP pour le transfert de pages Web, DNS pour les résolutions de noms, NFS pour le partage de fichiers, DHCP pour l'auto-configuration des machines, etc. C'est de loin la famille la plus riche, chaque application étant susceptible de disposer de son propre protocole.

CULTURE **Le modèle OSI**

Le modèle OSI tente de décrire de façon générique les empilements de protocoles réseau. Il comporte les couches physique (le matériel), d'accès au média, réseau, transport, session, présentation, et application, qu'on numérote parfois de 1 à 7.

Cela décrit bien les protocoles du monde IP jusqu'au niveau 4. Les couches session et présentation en sont en effet absentes, et leurs fonctionnalités gérées par les protocoles du niveau applicatif (couche 7).

CULTURE **Piles de protocoles**

On parle souvent de la pile IP pour désigner le monde des protocoles gravitant autour du protocole IP. Il existe d'autres piles (AppleTalk, IPX/SPX), mais elles tombent en désuétude à mesure que les applications migrent sur IP.

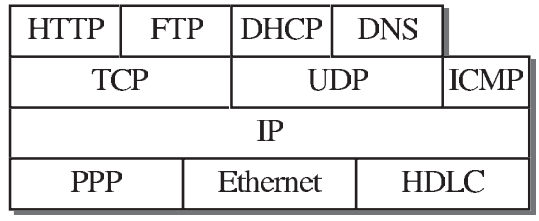


Figure 9–1 Pile de protocoles autour du protocole IP

Ces protocoles utilisent des protocoles de transport, essentiellement TCP ou UDP. UDP assure un transport non fiable (des données peuvent se perdre) ; TCP assure un transport fiable, mais plus coûteux en ressources. Tous deux utilisent un protocole réseau pour communiquer d'une machine à l'autre sur Internet : IP.

IP permet d'échanger des paquets entre machines de réseaux physiques différents. Il utilise pour cela divers protocoles d'accès au média, selon le réseau

accueillant la machine : Ethernet, PPP (liaison téléphonique ou spécialisée), PPPoE, PPTP, PPPoA (ADSL), ... La figure 9.1 donne un aperçu de l'empilement de quelques protocoles. Détaillons donc tout cela.

Ethernet

Panoplie des médias disponibles

Ethernet est la technologie la plus répandue pour les réseaux locaux. Au niveau physique, il se décline sous de nombreuses formes : câbles coaxiaux avec prises BNC, gros câbles (en anglais *thick Ethernet*) avec prise AUI, doubles paires torsadées avec prise RJ45, fibre optique, et même radio (*wireless Ethernet*). Tout est possible, et parfois à des vitesses différentes, telles que 10 Mb/s, 100 Mb/s, ou 1 Gb/s pour les versions sur fibre ou paires torsadées. La photo 9.2 présente les types de connectique les plus courants.

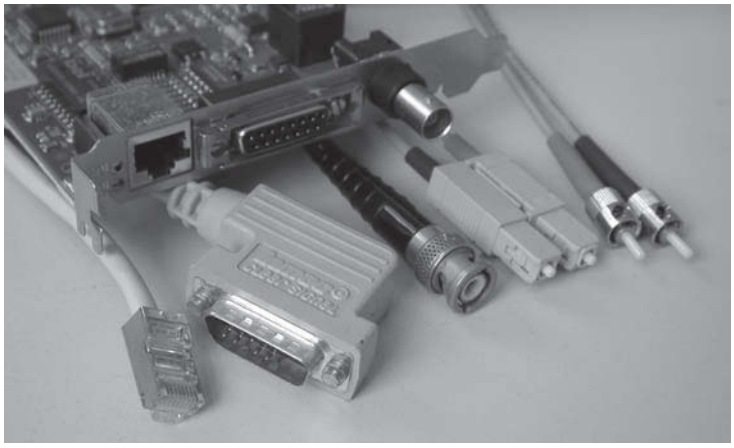


Figure 9–2 Quelques médias : carte Ethernet munie de trois connecteurs. De gauche à droite : RJ45, AUI et BNC, surplombant les câbles et les connecteurs correspondants. À droite, deux paires de fibres optiques avec des connecteurs différents

PLUS LOIN **Les dénominations d'Ethernet**

On utilise une dénomination compacte pour désigner la combinaison d'un média et d'une vitesse d'opération d'un réseau Ethernet. Exemples :

- 10base2 pour le câblage coaxial à prise BNC, à 10 Mb/s ;
- 10base5 pour les gros câbles à prise AUI, à 10 Mb/s ;
- 10base-T pour la paire torsadée à prise RJ45, opérée à 10 Mb/s ;
- 100base-T pour la paire torsadée à prise RJ45, opérée à 100 Mb/s ;
- 10base-F pour la fibre optique, opérée à 10 Mb/s.

Tout ce câblage se connecte aux cartes Ethernet des machines, parfois via un transcepteur (*transceiver* en anglais) s'il faut changer de média. Par exemple, pour brancher une carte avec connecteur AUI sur un câble à connecteur RJ45, il faudra utiliser un transcepteur 10base5 vers 10base-T.

La topologie des réseaux dépend du câblage : en bus pour le coaxial, en étoile pour les paires torsadées. Le centre de l'étoile est alors un concentrateur (*hub*, ou répéteur), dont le rôle consiste à répercuter sur tous ses ports toute trame (paquet d'informations) parvenant sur l'un d'entre eux. On peut relier des *hubs* les uns aux autres pour reproduire une topologie en étoile entre ces derniers, évitant ainsi de tirer trop de câble : pour relier deux bâtiments, on placera un *hub* dans chaque bâtiment, puis on les reliera entre eux.

On ne parle pas tous à la fois

Ethernet permet à plusieurs machines de partager le même média pour communiquer, grâce au protocole CSMA/CD (*Carrier Sense Multiple Access/Collision Detection*). Son fonctionnement est relativement simple : quand une machine désire émettre une trame, elle écoute si une autre machine est déjà en train de transmettre, auquel cas elle attend un temps aléatoire puis recommence. Quand la voie est libre, elle envoie sa trame. Mais il se peut qu'une autre machine ait envoyé une trame au même moment : on parle alors de collision Ethernet. Les machines détectent la collision, attendent un temps aléatoire, et tentent à nouveau d'émettre la trame.

Chaque trame contient une en-tête et un contenu, comme illustré à la figure 9.6. Le contenu est utilisé par le protocole de niveau supérieur, comme IP. L'en-tête contient deux adresses Ethernet, source et destination, ce qui permet à une machine de détecter qu'une trame lui est destinée ou non, et de répondre à l'expéditeur.

CSMA/CD fonctionne bien tant que toutes les machines n'essaient pas de transmettre en même temps. Plus les machines du réseau émettent, plus on compte de collisions. L'évolution ultime de l'augmentation du trafic est l'effondrement

PLUS LOIN **Câble croisé ou pas ?**

Le câblage par paires torsadées comporte en général quatre paires, dont seulement deux sont utilisées par Ethernet. La figure 9.3 donne le schéma de câblage d'une prise RJ45.

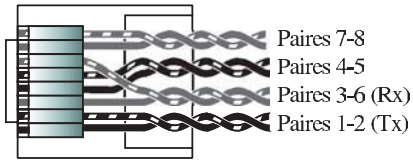


Figure 9–3 Câblage d'une prise RJ45 : attention à la paire 3-6, qui enjambe la paire 4-5

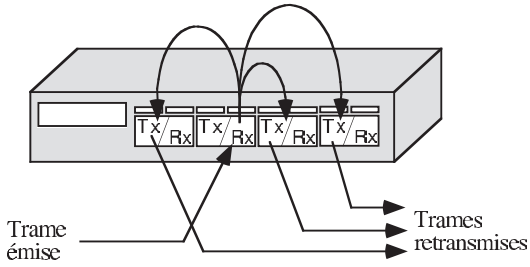


Figure 9–4 Fonctionnement d'un hub

Les machines utilisent une paire pour l'envoi (dite *Tx*) et une paire pour la réception (dite *Rx*). Les câbles ordinaires sont droits (la paire 1-2 mène à la paire 1-2 et la paire 3-6 mène à la paire 3-6) ; les prises des *hubs* sont donc croisées : l'appareil oppose un *Tx* au *Rx* de la machine, et inversement. Quand le *hub* reçoit une trame sur une paire *Rx*, il la renvoie sur les paires *Tx* de toutes les autres prises. La figure 9.4 illustre ce fonctionnement.

Un problème se pose lors du branchement direct de deux *hubs* ou de deux machines : les paires *Tx* se font alors face, et les trames envoyées n'atteignent pas la paire *Rx* de l'équipement opposé. On utilise donc des câbles croisés, qui échangent ces deux types de paires, comme on peut l'observer sur la figure 9.5.

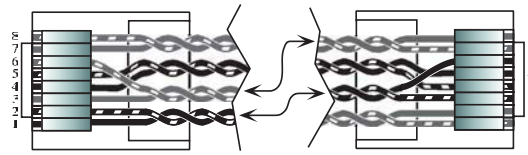


Figure 9–5 Câblage croisé : les paires 1-2 et 3-6 sont échangées entre les deux extrémités du câble.

Alternative : la plupart des *hubs* ont un port que l'on peut croiser ou décroiser à l'aide d'un bouton, ce qui permet de relier deux *hubs* avec un câble droit. Les équipements récents sont aussi capables de détecter la paire *Tx* opposée et de s'adapter, ce qui évite de se poser des questions sur les problèmes de croisements.

Ethernet, quand la cacophonie est telle qu'aucun système ne parvient à transmettre une trame entière. Autre souci : chaque machine voit passer toutes les trames, même celles qui ne lui sont pas destinées. Cela peut poser un problème de sécurité : il est facile d'intercepter le mot de passe d'un collègue qui lit son courrier électronique.

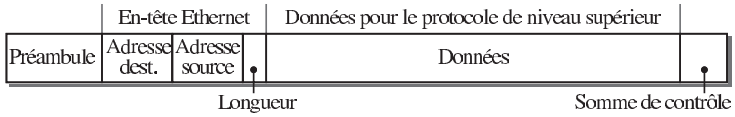


Figure 9–6 Format des trames Ethernet

Réseaux commutés

Ces deux problèmes se règlent en utilisant des commutateurs (*switches*) en lieu et place des concentrateurs (*hubs*). Le commutateur agit comme le concentrateur, mais il est capable de comprendre les en-têtes des trames. Lorsqu'il observe une trame sur un port, il enregistre l'adresse physique source et le numéro du port : il sait désormais que la machine correspondant à cette adresse physique se trouve sur ce port. Par conséquent, lorsqu'il devra transmettre une trame pour cette adresse, il se contentera d'émettre sur le bon port, contrairement au concentrateur, qui rediffuserait sur tous les ports.

Le commutateur permet des communications simultanées entre deux paires de machines, sans collision. Espionner le trafic destiné à son voisin devient plus difficile, puisqu'il n'est pas envoyé à tout le réseau, mais sur le seul port où se situe la machine visée.

Les réseaux équipés de commutateurs sont appelés réseaux commutés, et assurent de forts gains de performances pour un nombre élevé de machines. Ils ont en outre l'avantage de ne demander aucune configuration : il suffit de remplacer les concentrateurs par des commutateurs pour apprécier immédiatement la différence.

PLUS LOIN **Les adresses Ethernet**

On parle d'adresse Ethernet, MAC (*Medium Access Control*), physique, ou tout simplement d'adresse de niveau 2. Les adresses Ethernet, longues de 48 bits, sont souvent exprimées en hexadécimal, comme suit : `00:a0:40:24:74:cb`

Ethernet ne pourrait pas fonctionner sans la garantie d'unicité des adresses MAC. Chaque carte Ethernet a donc une adresse, attribuée à la fabrication, et dont on assure l'unicité en affectant à chaque fabricant un préfixe de 24 bits (`00:a0:40` est par exemple réservé à Apple). Le fabricant prend garde à produire des cartes dont les 24 derniers bits de l'adresse sont tous différents. C'est l'IEEE, à l'origine du standard Ethernet (normes IEEE 802.1 à 802.11), qui attribue les préfixes.

Configuration d'Ethernet

Le niveau Ethernet nécessite fort peu de configuration, la plupart des paramètres ajustables étant auto-négociés entre machines et concentrateurs ou commutateurs. Dans certaines situations, il est parfois souhaitable d'intervenir.

On peut par exemple configurer le média utilisé si l'on dispose d'une carte munie de plusieurs connecteurs qui pour une raison quelconque n'a pas pu détecter le connecteur réellement utilisé. Sous Unix, c'est la commande **`ifconfig`** qui permet cela. Pour choisir le connecteur AUI sur un système BSD, on tapera ainsi :

```
# ifconfig fxp0 media 10base5
```

ATTENTION **Mauvaise configuration**

Les erreurs de configuration laissant une interface en *full-duplex* et l'extrémité opposée du lien en *half-duplex* sont assez pernicieuses : le lien fonctionne, mais avec des performances lamentables.

C'est pourquoi il ne faut rien forcer en *full-duplex* ou *half-duplex* sans avoir vérifié la configuration de l'équipement opposé : on évitera ainsi de perdre du temps à diagnostiquer un incompréhensible problème de performances par la suite.

CULTURE **Changement d'adresse MAC**

Les stations Sun disposent d'une adresse MAC pour la machine, stockée en ROM, que toutes les cartes de la machine sont configurées pour utiliser au démarrage. On pourra ainsi changer de carte Ethernet sans changer d'adresse MAC.

SIMPLIFIEZ-VOUS LA VIE **Commutateurs administrables**

Dans le cas de commutateurs et concentrateurs administrables, il est également possible de forcer les ports à certaines configurations. C'est très utile lorsqu'ils sont raccordés à des matériels incapables d'auto-négocier correctement, ou qui ne peuvent pas être forcés dans une configuration particulière (c'est parfois le cas pour les imprimantes réseau).
Les équipements réseau administrables ont d'autres avantages considérables. Ils peuvent indiquer sur quel port se trouve la machine utilisant une adresse MAC donnée, et donner les statistiques d'utilisation des ports, ce qui mettra en lumière toute situation anormale. De tels équipements sont donc recommandés ; tôt ou tard, on ne pourra que se féliciter d'en avoir fait l'acquisition.

La vitesse d'opération d'Ethernet est elle aussi auto-négociée entre la machine et le concentrateur ou commutateur. Si tous deux sont capables de fonctionner à 100 Mb/s, le lien sera exploité à cette vitesse. Il sera en 10 Mb/s dans le cas contraire. Quand l'auto-négociation se passe mal, ou si elle n'est pas disponible (c'est par exemple le cas sur fibre optique), on peut forcer une interface à 10 ou 100 Mb/s :

```
# ifconfig fxp0 media 10baseT
# ifconfig fxp0 media 100baseTX
```

Les communications bidirectionnelles simultanées (*full-duplex*) ou alternées (*half-duplex*) sont en principe auto-négociées, mais cette négociation échoue souvent. Il faut alors configurer à la main.

```
# ifconfig fxp0 media 10baseT-FDX ❶
# ifconfig fxp0 media 10baseT ❷
```

❶ Interface forcée en *full-duplex*

❷ Interface forcée en *half-duplex*

Dernier élément paramétrable : l'adresse MAC elle-même. Certaines cartes le permettent, et cette opération fera par exemple accepter une nouvelle carte par un serveur DHCP configuré pour n'attribuer d'informations de connexion qu'aux adresses MAC qu'il connaît déjà. Ainsi, l'adresse MAC n'est pas liée de façon irrémédiable à une carte : il est possible d'en usurper une autre.

Internet

Internet est une interconnexion de réseaux hétérogènes : Ethernet, *Token ring*, liaisons point à point en PPP ou HDLC, réseaux étendus FDDI, ATM, *Frame-Relay*, et bien d'autres encore.

Au sein d'un réseau donné, la communication est simple : il suffit d'émettre des trames, que le réseau se chargera d'acheminer à bon port. La difficulté est d'échanger des données entre réseaux de types différents.

Si les trames sont semblables, comme dans le cas de deux réseaux Ethernet, on peut construire un *bridge*, mais cette solution n'est pas toujours viable. Les protocoles de découverte dynamique du réseau doivent envoyer des trames à toutes les machines (*broadcasts*). Les *bridges* relaient ces trames, et dans le cas d'un réseau étendu, on obtiendra rapidement un effondrement Ethernet, même avec des commutateurs. Pour un réseau de grande envergure, il faut donc trouver une autre solution. De plus, dans de nombreux cas, on ne pourra pas traduire les en-têtes de niveau 2 entre deux réseaux, si par exemple les formats des adresses physiques diffèrent – comme entre un réseau Ethernet et un lien PPP.

CULTURE **Bridges**

Un *bridge* (pont) est un équipement qui dispose en général d'un faible nombre de ports et raccorde deux réseaux pouvant différer légèrement (comme Ethernet et *Token ring*). Le *bridge* jouera le rôle d'un commutateur entre les deux réseaux. Si les réseaux diffèrent, il pourra de plus traduire les en-têtes de niveau 2 pour que les paquets puissent passer d'un réseau à l'autre.

Un protocole pour tout interconnecter

Il faut donc interconnecter les réseaux, au niveau local comme étendu. C'est IP (*Internet Protocol*) qui permet cela : il utilise les protocoles d'accès au média pour transmettre des paquets IP renfermant une en-tête IP et des données. Lorsqu'il faut communiquer d'un réseau à un autre, on installe un routeur IP.

Lorsque le routeur reçoit une trame de niveau 2 sur un port, il en extrait le paquet IP. Les informations contenues dans l'en-tête IP lui permettent de décider du port sur lequel ce paquet doit être transmis. Le paquet IP est alors replacé dans une trame de niveau 2 correspondant au réseau sous-jacent, et il est expédié sur le port adéquat. La figure 9.7 décrit ce processus.

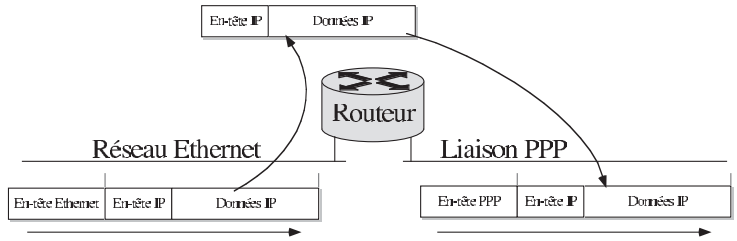


Figure 9–7 Fonctionnement d'un routeur

La commande **`traceroute`** permet d'explorer tous les routeurs par lesquels passent les paquets envoyés à une machine donnée :

```
$ /usr/sbin/traceroute www.freebsd.org
traceroute to www.freebsd.org (216.136.204.117), 30 hops max, 40 byte packets
 1  sf-guest.orsf2.pf.isc.org (204.152.184.194)  0.407 ms  1.381 ms  0.386 ms
 2  bas1-m.pao.yahoo.com (198.32.176.135)  3.353 ms  1.580 ms  1.389 ms
 3  ge-0-2-0.msr1.sc5.yahoo.com (216.115.100.233)  4.484 ms  5.531 ms  7.020 ms
 4  vl47.bas1-m.sc5.yahoo.com (66.163.160.230)  6.400 ms  2.790 ms  2.134 ms
 5  www.freebsd.org (216.136.204.117)  4.507 ms  3.434 ms  7.181 ms
$
```

`traceroute` sonde trois fois chaque routeur traversé. Les temps reportés en fin de ligne correspondent aux temps d'aller/retour de chaque sonde et de sa réponse.

L'en-tête IP contient au minimum 20 octets d'information, auxquels on peut ajouter des options. Voici la définition de l'en-tête IP, extraite de la RFC 791, qui définit le standard IP :

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

CULTURE **Routeur**

Un routeur est un équipement réseau disposant en général d'un faible nombre de ports, parfois prévus pour des réseaux différents. Il peut par exemple disposer de deux ports Ethernet et d'un port série pour se connecter à une liaison spécialisée.

Le rôle du routeur est de router les paquets en les dirigeant vers le bon réseau. Un routeur IP est capable de décider sur quelle interface un paquet doit être transmis en se fondant sur son adresse IP de destination.

On trouve des routeurs pour d'autres protocoles réseau : on peut par exemple router des paquets AppleTalk avec un routeur AppleTalk. Le paquetage Netatalk transformera une machine Unix en routeur AppleTalk.

CULTURE **Les RFC**

Les RFC (*Request For Comments*, ou appels à commentaires) sont les documents définissant les propositions de standards et les normes de l'Internet. Publiés sous forme de texte brut avec fins de lignes au format Unix, ils sont disponibles sur différents serveurs, et notamment à l'adresse : ftp://ftp.jussieu.fr/pub/rfc/rfc.

Quelques dizaines de RFC sont disponibles en version française à l'adresse suivante : http://abcdrfc.free.fr.

Les champs les plus importants sont sans aucun doute les adresses IP source et destination. Longues de 32 bits, elles peuvent s'exprimer en décimal comme ceci : `213.244.11.247`. L'adresse source permet à une machine recevant un paquet de répondre à l'émetteur ; l'adresse destination est utilisée par les routeurs pour faire les décisions de routage et décider du port où transmettre un paquet. Pour cela, le routeur s'appuie sur une table de routage.

Masques de sous-réseau et passerelle par défaut

Les machines non routeurs ont des tables de routage très réduites. Elles connaissent les machines accessibles directement (le sous-réseau) et celles (le reste d'Internet) pour lesquelles il faut envoyer les données à un routeur, également appelé passerelle par défaut.

Ce sont l'adresse IP et le masque de sous-réseau qui permettent à une machine de déterminer quelles adresses lui sont accessibles directement, Le masque de sous-réseau est une quantité de 32 bits, en général exprimée en décimal, comme l'adresse IP : `255.255.255.240`. On le trouve parfois exprimé en hexadécimal : `0xFFFFFFF0`.

Pour l'utiliser, il faut exprimer le masque de sous-réseau en binaire. Il prend alors toujours la forme d'un certain nombre de bits à 1 suivis de bits à 0. Tout autre format est invalide. Exemple :

Décimal	Binaire
255.255.255.128	11111111.11111111.11111111.10000000

VOCABULAIRE **Passerelle par défaut**

Dans les nombreux logiciels en anglais que vous rencontrerez, la passerelle par défaut sera appelée *default gateway* ou tout simplement *gateway*.

EN PRATIQUE **Les sous-réseaux**

Il est primordial de savoir calculer des masques de sous-réseaux si l'on souhaite diriger les évolutions du plan d'adressage d'un réseau. Le découpage en sous-réseaux permet une gestion plus rationnelle des flux, mais c'est aussi un point de passage obligatoire pour le cloisonnement d'un réseau à des fins de sécurité.

CULTURE **Attribution des adresses IP**

Le même problème se pose que pour les adresses MAC : il faut que les adresses IP soient uniques sur tout l'Internet. L'espace d'adresses IP est géré par l'IANA (*Internet Assigned Numbers Authority*).

L'IANA répartit les blocs d'adresses IP entre quatre entités régionales, qui les distribuent ensuite aux opérateurs et aux fournisseurs d'accès Internet :

- L'APNIC (*Asia Pacific Network Information Centre*) pour la région Asie/Pacifique ;
- L'ARIN (*American Registry for Internet Numbers*) pour l'Amérique du Nord et l'Afrique du Sud ;
- Le LACNIC (*Latin-American and Caribbean Internet Addresses Registry*) pour l'Amérique du Sud ;
- Le RIPE (*Réseaux IP Européens*) pour l'Europe, le Proche-Orient, l'Asie centrale, et l'Afrique du Nord.

Les opérateurs et fournisseurs d'accès distribuent ensuite les adresses à leur clients.

On peut consulter l'allocation des blocs d'adresses par l'IANA dans le document suivant : http://www.iana.net/assignments/ipv4-address-space. L'allocation régionale sera fournie par les serveurs **`whois`** concernés. Exemple pour l'ARIN :

```
$ whois -h whois.arin.net 17.112.152.32

OrgName:    Apple Computer, Inc.
OrgID:      APPLEC-3
Address:    20740 Valley Green Drive, MS32E
City:       Cupertino
StateProv:  CA
PostalCode: 95014
Country:    US

NetRange:   17.0.0.0 - 17.255.255.255
CIDR:       17.0.0.0/8
NetName:    APPLE-WWNET
NetHandle:  NET-17-0-0-0-1
Parent:
NetType:    Direct Assignment
NameServer: NSERVER.APPLE.COM
NameServer: NSERVER2.APPLE.COM
NameServer: NSERVER.EURO.APPLE.COM
NameServer: NSERVER.ASIA.APPLE.COM
Comment:
RegDate:    1990-04-16
Updated:    2000-05-23

TechHandle: ZA42-ARIN
TechName:   Apple Computer, Inc.
TechPhone:  +1-408-974-7777
TechEmail:  Apple-NOC@apple.com
```

On parle de masque car les bits à 1 indiquent quelle partie de l'adresse IP correspond au préfixe commun à toutes les machines du sous-réseau. Voici par exemple l'adresse IP et le masque affectés à l'interface d'une machine :

Décimal	Binaire
10.0.12.215	00001010.00000000.00001100.11010111
255.255.255.128	11111111.11111111.11111111.10000000

Et voici les adresses du sous-réseau correspondant :

Décimal	Binaire
10.0.12.128–255	00001010.00000000.00001100.1xxxxxxx

ALTERNATIVE **Notation compacte des masques de sous-réseau**

Les masques de sous-réseau commençant toujours par une série de 1 suivie d'une série de 0, on peut les résumer à leur nombre de 1. Ceci amène à une notation réduite, fréquente, où `255.255.255.128` s'écrit `/25`, car ce masque compte 25 bits à 1. On indique ainsi une adresse IP et son masque de façon assez compacte : `10.0.12.215/25`. C'est la notation « CIDR » (*Classless Inter-Domain Routing*, ou routage inter-domaines sans classe).

Interaction entre IP et Ethernet : ARP

Dans le cas des réseaux Ethernet, qu'une machine ait décidé d'envoyer un paquet IP directement à sa destination ou à un routeur, le problème reste le même : il faut envoyer un paquet à une machine dont on connaît l'IP, mais pas l'adresse physique. Or, il faut connaître l'adresse physique d'une machine pour pouvoir lui envoyer une trame Ethernet.

Un petit protocole intercalé entre Ethernet et IP permet de résoudre le problème : ARP (*Address Resolution Protocol*, ou protocole de résolution d'adresses). Il utilise une table, présente sur chaque machine utilisant Ethernet : la table ARP. Sous Unix, on peut en consulter le contenu avec la commande **`arp`** :

```
$ arp -a
manganese.local (192.168.3.1) at 00:c0:4f:44:b0:54 on ex0
mmX.local (192.168.3.39) at 00:02:55:f7:79:18 on ex0
mm6.local (192.168.3.46) at (incomplete) on ex0
mm22.local (192.168.3.62) at (incomplete) on ex0
in4.local (192.168.3.74) at (incomplete) on ex0
in18.local (192.168.3.88) at 00:c0:4f:82:5b:1c on ex0
? (192.168.3.255) at (incomplete) on ex0
```

Lors de l'envoi d'un paquet, le système consulte la table ARP à la recherche de l'adresse physique correspondant à une adresse IP donnée. S'il ne l'y trouve pas, il émet alors une requête ARP : il s'agit d'une trame envoyée à toutes les machines du réseau Ethernet, posant la question : quelle adresse physique correspond à telle adresse IP ? La machine concernée répond, et le système stocke cette information dans la table ARP. Il est possible d'observer l'activité ARP avec un outil comme **`tcpdump`** :

```
# tcpdump -n 'arp'
15:26:03.840285 arp who-has 10.0.12.2 tell 10.0.12.8
15:26:03.841548 arp reply 10.0.12.2 is-at 0:0:94:5:c9:51
```

Quand il dispose de l'adresse physique du destinataire ou du routeur, il suffit au système de lui expédier le paquet IP dans une trame Ethernet. L'adresse IP de destination sera de toutes façons celle de la destination finale, même s'il faut passer par un routeur.

SÉCURITÉ **Renifleurs**

Les outils permettant de capturer le trafic réseau s'appellent des renifleurs (*sniffers*). **`tcpdump`** est le plus classique ; on le trouve dans la distribution de base de beaucoup de systèmes Unix. Il y en a d'autres : Ethereal, avec une interface graphique, et Snort, système de détection d'intrusion que l'on peut utiliser comme renifleur. Ethereal et Snort sont tous deux disponibles dans les systèmes de paquetages des BSD. Certains Unix disposent aussi de renifleurs spécifiques, tels que **`snoop`** sous Solaris.

Seul root a le droit d'utiliser un renifleur, pour des raisons de sécurité assez évidentes : beaucoup de protocoles faisant encore passer les mots de passe en clair sur le réseau, un utilisateur maniant un renifleur sur un serveur pourrait rapidement récupérer les mots de passe de tous ses collègues.

CULTURE **Cousins d'ARP**

Les autres piles réseau ont le même problème qu'IP pour faire la correspondance entres les adresses de niveau 3 et les adresses Ethernet. Elles disposent pour cela d'un protocole similaire à ARP ; dans le cas de la pile AppleTalk, il s'appelle AARP (*AppleTalk Address Resolution Protocol*, protocole de résolution d'adresses AppleTalk).

Fragmentation

IP doit pouvoir faire traverser à ses paquets des réseaux très différents, en particulier au niveau des tailles maximales des trames. Sur Ethernet, le MTU (*Maximum Transmission Unit*) est de 1500 octets. Sur une liaison SLIP (alternative à PPP pour les liens série), il vaut 296 octets. Il s'agit de limitations propres à chaque type de réseau, on ne peut les éviter.

VOCABULAIRE **MTU et PMTU**

Le MTU (*Maximum Transmission Unit*, ou unité de transmission maximale) est la taille maximale des trames que peut transporter un réseau. C'est une limitation structurelle avec laquelle il faut composer. Tout paquet plus gros que le MTU devra être fragmenté en plusieurs trames. La commande **`ifconfig`** indique le MTU de chaque interface :

```
$ ifconfig sl0
sl0: flags=c010<POINTOPOINT,LINK2,MULTICAST> mtu 296
```

Le PMTU (*Path Maximum Transmission Unit*, ou unité de transmission maximale du chemin) est le plus petit MTU de tous les réseaux présents sur un chemin donné. C'est la taille maximale d'un paquet qui ne subira pas de fragmentation.

IP gère cette difficulté en fragmentant les paquets. Ces fragments ne sont en principe rassemblés qu'à destination, et c'est à la machine destinataire qu'incombe cette tâche.

Cette règle admet des exceptions : sous OpenBSD, PacketFilter propose une fonction de renormalisation pour rassembler les fragments des paquets IP sur un routeur intermédiaire. IPTables et IPChains sous GNU/Linux proposent une fonctionnalité similaire.

Configuration IP

Le cas le plus simple de configuration IP est celui d'une machine ne comptant qu'une interface réseau : il suffit alors d'indiquer l'adresse IP et le masque, ainsi que la passerelle par défaut.

Cela peut se compliquer à loisir ; il est possible de compter plusieurs interfaces, disposant chacune de leurs propres adresses et masques de sous-réseau. On peut même attribuer plusieurs adresses à une même interface, chacune avec son propre masque de sous-réseau, ou configurer plusieurs interfaces avec la même adresse IP, si un *bridge* les relie.

La configuration peut être manuelle, ou assurée par un protocole de configuration dynamique. Par exemple, lors d'une connexion PPP sur le réseau téléphonique, le serveur PPP distant fournit les informations de configuration pour l'interface PPP : adresse IP, masque de sous-réseau, et passerelle par défaut. Comme on l'a vu au chapitre 7, des extensions que l'on doit à Microsoft permettent parfois d'obtenir aussi la configuration du DNS.

Autres possibilités de configuration dynamique : RARP (*Reverse Address Resolution Protocol*), qui consiste à envoyer un paquet en *broadcast* Ethernet, en demandant quelle est sa propre adresse IP. Un serveur RARP répond par un paquet

PIÈGE À C... **Les configurations non fonctionnelles**

Quand on dispose de plusieurs cartes réseau, on peut imaginer de les placer toutes sur le même sous-réseau pour répartir la bande passante entre les différentes interfaces. Inutile d'essayer, cela ne fonctionne pas.

Autre erreur à ne pas commettre : la passerelle par défaut doit être accessible directement, donc se trouver dans le même sous-réseau qu'au moins une des adresses IP de la machine. Sans cela, il sera impossible de quitter le sous-réseau.

CULTURE **PPP**

PPP est un protocole de communication point à point (*Point to Point Protocol*), principalement utilisé pour se connecter à Internet avec un modem, via le réseau téléphonique. On peut aussi l'employer pour faire communiquer en IP deux machines reliées par un câble série. Sur les systèmes BSD et un certain nombre de systèmes Unix, c'est le *daemon* **`pppd`** qui joue à la fois le rôle de client ou de serveur PPP.

contenant cette information. Cela ne fournit pas le masque de sous-réseau ni la passerelle par défaut, qu'il faudra découvrir autrement. BootP et son évolution DHCP sont des protocoles d'auto-configuration plus complets fonctionnant sur Ethernet. Ils permettent d'indiquer toute la configuration IP, les paramètres du DNS, un serveur de *netboot*, un maître NIS... Beaucoup de choses sont possibles, et pour ce qui n'est pas prévu, DHCP prévoit un mécanisme d'extensions pour transmettre à peu près ce que l'on souhaite.

Adresses spéciales

Terminons la visite du protocole IP avec quelques adresses IP spéciales, fréquemment rencontrées.

Adresse nulle

L'adresse `0.0.0.0` est parfois utilisée en l'absence d'adresse véritable. Par exemple, les requêtes DHCP émises par une machine qui cherche son adresse IP ont pour adresse IP source `0.0.0.0`.

Adresse de réseau et adresse de *broadcast*

Chaque sous-réseau réserve deux adresses spéciales, qu'on ne peut attribuer à aucune machine : il s'agit de la première et de la dernière adresse du sous-réseau, respectivement adresse du réseau et adresse de *broadcast* (diffusion). Cette dernière permet d'envoyer un paquet à toutes les machines du réseau.

Pour l'exemple `10.0.12.153/25`, l'adresse de réseau est `10.0.12.128` et l'adresse de *broadcast*, `10.0.12.255`.

On rencontre parfois l'adresse de *broadcast* `255.255.255.255`, qui permet de s'adresser à tout le réseau local sans avoir encore obtenu d'adresse IP.

Adresses locales

Toute la plage `127.0.0.0/8` est réservée à l'usage local de chaque machine. Aucun paquet IP ne devrait transiter sur un réseau avec l'une de ces adresses comme source ou comme destination.

Cette plage est souvent extrêmement sous-employée : en général, seule l'adresse `127.0.0.1` est configurée. C'est l'adresse locale, qui permet à la machine de se connecter à elle-même sans passer par le réseau, ce qui est nécessaire pour de nombreux programmes (on parle d'interface de boucle de retour, ou *loopback*).

EN PRATIQUE **Adresses privées**

Les adresses privées sont très appréciées sur le plan de la sécurité, car il est impossible d'y accéder directement depuis Internet : elles fournissent donc de bons gages de sécurité (attention, elles restent toutefois accessibles par rebonds). Leur emploi pose un certain nombre de problèmes avec les applications qui prévoient justement une connexion directe de l'extérieur sur la machine.
Citons notamment : FTP, *X Window System*, RealPlayer, ICQ, et tous les logiciels de communication de pair à pair (*Peer to Peer*).

PLUS LOIN **Options IP et IPSec**

IP propose un certain nombre d'options rarement utilisées. Certaines, de réputation assez sulfureuse, ont été à l'origine de problèmes de sécurité. C'est le cas de l'option source-route, qui permet de préciser un routage des paquets prévalant sur les tables de routage normales. Ceci ouvrant la porte à de nombreuses techniques de contournement, on préfère souvent filtrer ces options ; nous verrons comment au chapitre 10.
D'autres options sont au contraire plutôt désirables du point de vue de la sécurité. C'est le cas des options de chiffrement et d'authentification, utilisées par IPSec. Elles établissent une sécurité au niveau IP qui permettra d'utiliser **`telnet`** pour se connecter à une machine distante sans plus rien craindre des renifleurs. Pour découvrir comment mettre en œuvre IPsec sur un système BSD, reportez-vous à la page **`man`** de **`ipsec`**.

CULTURE **Nombre d'adresses au mètre carré**

Les chimistes remarqueront avec amusement que le nombre d'adresses IPv6 au mètre carré sur Terre est supérieur à la constante d'Avogadro (pour ceux qui n'ont pas fait de chimie, il s'agit d'une quantité énorme indiquant le nombre d'atomes présents dans un gramme d'hydrogène).

Adresses privées

Dans certaines situations, il faut monter un réseau sans avoir reçu d'adresse d'un fournisseur d'accès : c'est par exemple le cas si l'on monte un réseau privé chez soi. On peut aussi avoir besoin de plus d'adresses que le fournisseur n'en a fourni. La solution, ce sont les adresses privées.

Trois plages sont réservées à l'usage privé : `10.0.0.0/8`, `172.16.0.0/12`, et `192.168.0.0/16`. Tout le monde peut les employer, mais elles ne doivent pas être utilisées pour communiquer sur Internet. En utilisant ces plages, on a l'assurance de ne pas entrer en conflit avec un autre réseau. Il faut alors avoir recours à des dispositifs comme la translation d'adresses (*Network Address Translation* en anglais, ou NAT) ou des serveurs mandataires (*proxies*) pour communiquer avec l'extérieur. Nous aborderons ces problèmes au chapitre 10.

Classes d'adresses

Le plan de découpage initial des adresses IP était assez maladroit. Il définissait de façon définitive un certain nombre de blocs d'adresses, appelés classes, de taille fixée. En voici les définitions :

`0.0.0.0` – `127.255.255.255`	128 classes A en `/8` (16777216 adresses par classe)
`128.0.0.0` – `191.255.255.255`	16 384 classes B en `/16` (65536 adresses par classe)
`192.0.0.0` – `223.255.255.255`	2 097 152 classes C en `/24` (256 adresses par classe)
`224.0.0.0` – `239.255.255.255`	268 millions d'adresses de classe D, réservées au trafic *multicast*.
`240.0.0.0` – `255.255.255.255`	268 millions d'adresses réservées à un usage futur.

Le routage inter-domaines sans classe (CIDR : *Classless Inter-Domain Routing*) a rendu les classes d'adresses obsolètes, mais il est bon de savoir à quoi elles correspondent, car on entend encore parler de « classe C » pour designer un réseau `/24`.

Évolution vers IP version 6

La grande limitation de la version actuelle d'IP (IP version 4, ou IPv4), c'est le nombre d'adresses, et surtout la manière dont elles ont été initialement distribuées : quelques entreprises, comme HP, Xerox, ou Dupont de Nemours, se sont vu attribuer des classes A. Ces attributions d'adresses par blocs de 16 millions paraissent aujourd'hui indécentes : on n'offre désormais aux nouveaux venus que des blocs de taille inférieure à des classes C.

IP version 6 doit résoudre tout cela en proposant des adresses longues de 128 bits. Le nouveau nombre d'adresses maximal théorique, 340282366920938463463374607431768211456, devrait satisfaire tout le monde pour assez longtemps.

Les adresses IPv6 s'expriment en hexadécimal, par paquets de 16 bits. Exemple : `fe80:0000:0000:0000:2a0:40ff:fe24:74cb`. Pour les raccourcir un peu, on peut remplacer les séries de zéros par un double : (deux points) comme ceci : `fe80::2a0:40ff:fe24:74cb`. Bien entendu, on n'a le droit d'utiliser ce raccourci qu'une seule fois dans l'adresse.

Migration vers IPv6

La migration vers IPv6 se fera progressivement. Un certain nombre d'outils savent faire coexister IPv4 et IPv6 : le but est de faire passer des paquets IPv4 dans des réseaux IPv6, des paquets IPv6 dans des réseaux IPv4, ou de faire communiquer des machines IPv4 et IPv6.

Des utilitaires permettent ainsi de mettre en place des tunnels IPv4 dans IPv6, des tunnels IPv6 dans IPv4, des translateurs d'adresses de IPv4 vers IPv6... Tous ces outils sont disponibles dans les systèmes BSD. Pour les essayer, consultez les pages **man** de **gif**, **faith**, et **faithd**.

B.A.-BA **Encapsulation, tunnels**

L'encapsulation de protocoles consiste à placer le paquet d'un protocole dans un paquet d'un autre protocole. TCP est habituellement encapsulé dans IP, lui-même encapsulé dans les protocoles de niveau 2.

Quand on encapsule IP dans un autre protocole de réseau, on met en place un tunnel. Les machines situées aux deux extrémités du tunnel ont l'impression de ne traverser que deux routeurs IP entre l'encapsulation et la décapsulation, mais ces derniers peuvent être très éloignés l'un de l'autre, dans un réseau composé de dizaines de routeurs non IP.

PLUS LOIN **Adresses multicast**

Le service *multicast* permet de diffuser efficacement des paquets à plusieurs machines : ils ne seront pas dupliqués tant que cela ne sera pas nécessaire. Les machines intéressées par un certain trafic s'abonnent à une adresse *multicast* via le protocole IGMP (*Internet Group Management Protocol*, ou protocole de gestion de groupe sur Internet). Les routeurs *multicast* sont capables de s'abstenir d'envoyer sur une interface donnée des paquets *multicast* pour lesquels aucune machine du réseau n'est abonnée. La figure 9.8 et la figure 9.9 illustrent la différence entre les diffusions sans et avec *multicast*.

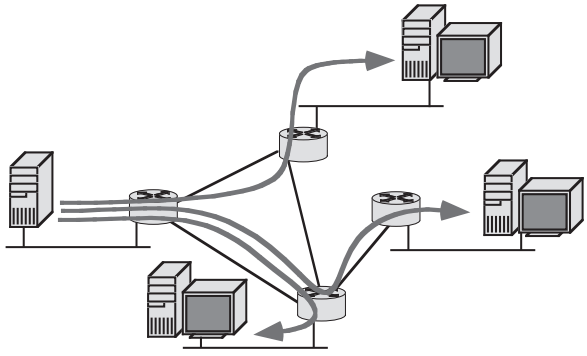


Figure 9–8 Flux vidéo lors d'une diffusion sans multicast : le serveur émet autant de flux qu'il y a de clients

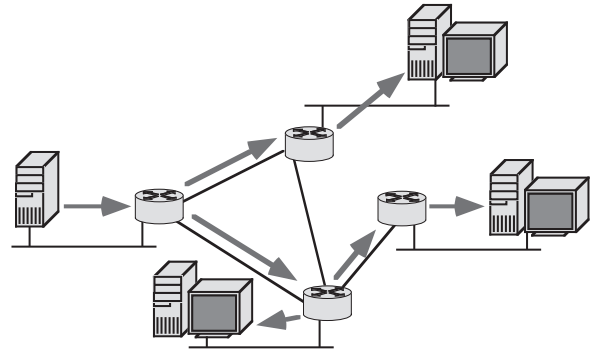


Figure 9–9 La même chose en multicast : un seul flux est émis, et c'est aux routeurs de le subdiviser autant qu'il le faut.

Le *multicast* a un certain nombre d'applications, dont les radios et télévisions en ligne, ainsi que les systèmes de conférence. Les machines bénéficiant du service *multicast* (c'est-à-dire celles qui sont reliées à des routeurs *multicasts*) constituent un ensemble baptisé MBone.

Pour monter un routeur *multicast* sous BSD, reportez-vous à la page **man** de **mrouted**. Pour tester des applications tirant parti du trafic *multicast*, essayez des outils tels que **rat**, **sdr**, ou **vic**, disponibles dans les systèmes de paquetages des trois BSD.

Autres protocoles réseau

IP n'est pas le seul protocole réseau. Ethernet étant exploitable simultanément par plusieurs protocoles réseau à la fois, vous découvrirez peut-être, au détour du réseau local, des paquets DDP (couche réseau de la pile AppleTalk), IPX (pile IPX/SPX de Novell), NetBEUI (NetBIOS fonctionnant directement sur Ethernet sans utiliser IP ; une vieillerie mise au point par IBM puis utilisée par Novell et Microsoft), ...

Tous ces protocoles, bien distincts d'IP, fournissent des fonctionnalités similaires, avec des adresses de formats différents. Ils sont en voie de disparition, les protocoles applicatifs qui les utilisent ayant pour la plupart migré sur TCP/IP. On les rencontre parfois sur d'anciennes machines.

Un protocole de contrôle pour IP

L'exploitation d'un réseau IP impose l'échange régulier de messages de contrôle entre les machines pour signaler des conditions exceptionnelles ou effectuer des diagnostics. Ces messages sont acheminés par le protocole ICMP (*Internet Control Messages Protocol*, ou protocole de messages de contrôle pour Internet), encapsulé dans IP.

Les paquets ICMP commencent par un type et un code de 8 bits chacun, qui les caractérisent. La suite dépend du type de message. Par exemple, un message ICMP de type 3 est renvoyé lorsqu'un paquet ne peut pas être amené à sa destination. Le code indique s'il s'agit d'un réseau inaccessible, d'une machine inaccessible, d'une option source-route qui a échoué... Le reste du message ICMP inclut une copie de l'en-tête IP du paquet original pour aider la machine émettrice à comprendre ce qui s'est passé.

Autre exemple, les messages ICMP de type 4, dits *ICMP source quench*, indiquent à la machine émettrice de ralentir ses émissions. Ils sont utilisés pour le contrôle de flux, et éviter l'engorgement des réseaux. Certains administrateurs filtrent tout ICMP, ce qui annule malheureusement ces mécanismes de contrôle de flux.

Dernier exemple : les paquets ICMP de types 8 (*echo request*), et 0 (*echo reply*) servent pour la commande **`ping`**. En utilisant **`ping`** pour tester la connectivité d'une machine, on envoie des ICMP de type 8, et la machine testée répond avec des ICMP de type 0.

PLUS LOIN **Application ICMP : fonctionnement de `traceroute`**

La commande **`traceroute`** permet d'obtenir la liste des routeurs traversés pour atteindre une machine. Cette commande fonctionne en exploitant le champ TTL (*Time To Live*) du protocole IP, qui évite aux paquets de déambuler indéfiniment en cas d'erreur de routage : à chaque traversée de routeur, le TTL est décrémenté. Quand il atteint zéro, le routeur détruit le paquet original et renvoie un paquet ICMP de type 11 et de code 0, signalant l'expiration du TTL.

`traceroute` envoie d'abord un paquet UDP de TTL 1, refoulé au premier routeur, dont il reçoit un message ICMP l'informant de l'incident. Le deuxième paquet UDP, de TTL 2, permet de découvrir le deuxième routeur, et ainsi de suite.

La trace indique parfois des astérisques : le paquet de sondage ou le paquet ICMP de réponse a alors été filtré ou perdu.

Protocoles de transport : TCP et UDP

IP ne fait qu'assurer l'interconnexion entre les réseaux. Ceci ne comprend que la gestion du routage et de la fragmentation des paquets trop volumineux pour passer. IP n'offre aucune garantie sur la qualité du service : les paquets peuvent être perdus, dupliqués, désordonnés, voire corrompus. Cette carence de garanties a une explication simple : tous les réseaux utilisant IP n'offrent pas les mêmes

garanties ; IP doit donc aligner ses prétentions sur le plus petit dénominateur commun.

Autre limitation d'IP : il ne gère que des communications de machine à machine. Rien n'est prévu pour trier des communications en fonction de leur processus de destination sur la même machine.

TCP (*Transfer Control Protocol*, ou protocole de contrôle de transfert) vise à résoudre ces deux problèmes. Un autre protocole, UDP (*User Datagram Protocol*), ne s'attaque lui qu'au problème de multiplexage des communications, et pas à leur fiabilisation. Nous verrons plus loin que contrairement à ce que l'on pourrait imaginer, il est préférable à TCP dans certaines situations.

PLUS LOIN **Les types ICMP**

On a rarement besoin de l'ensemble des types ICMP, mais les curieux seront heureux de lire ce qui suit.

Type	Code	Description
0	0	*Echo reply* : réponse à un **`ping`**.
3	0	*Net unreachable* : le réseau destination est inaccessible.
3	1	*Host unreachable* : la machine destination est inaccessible.
3	2	*Protocol Unreachable* : la machine destination ne reconnaît pas le protocole de transport utilisé.
3	3	*Port unreachable* : le port destination (généralement UDP) n'est pas lié à un processus.
3	4	*Fragmentation needed and DF set* : le paquet envoyé portait le drapeau DF (*Don't Fragment* : ne pas fragmenter), mais était trop gros pour atteindre sa destination sans être fragmenté.
3	5	*Source route failed* : le paquet portait l'option IP *source route*, qui permet d'indiquer un routage différent de celui des tables de routage, et ce routage a échoué.
4	0	*Source quench* : message de régulation de contrôle de flux, envoyé à une machine pour lui demander de ralentir son émission.
5	0	*Redirect datagrams for the network* : message indiquant un changement de routage pour un réseau entier.
5	1	*Redirect datagrams for the host* : message indiquant un changement de routage pour une machine.
5	2	*Redirect datagrams for the type of service and the host* : message indiquant un changement de routage pour un réseau entier et pour un type de service donné.
5	3	*Redirect datagrams for the type of service and the network* : message indiquant un changement de routage pour une machine et pour un type de service donné.
8	0	*Echo request* : envoi d'un **`ping`**.
11	0	*Time to live exceeded in transit* : paquet envoyé lorsqu'un paquet IP est détruit par un routeur à cause de son TTL tombé à zéro (voir à ce sujet l'aparté exposant le fonctionnement de **`traceroute`**).
11	1	*Fragment reassembly time exceeded* : signale la destruction d'un paquet IP faute d'avoir reçu tous ses fragments à temps.
12	0	*Parameter problem* : indique une erreur dans l'en-tête IP.
13	0	*Timestamp* : demande de l'envoi de l'heure d'une machine.
14	0	*Timestamp reply* : réponse à une demande d'heure.
15	0	*Information request* : demande d'informations de configuration.
16	0	*Information reply* : réponse à une demande d'informations de configuration. Une machine peut découvrir son adresse IP par ce procédé (qui reste assez rare ; on lui préfère DHCP).

Des connexions fiables avec TCP

Le rôle de TCP est d'utiliser IP pour fournir un service fiable, offrant les garanties suivantes : les paquets ne sont ni perdus, ni dupliqués ; ils arrivent dans l'ordre d'émission ; ils ne sont pas corrompus. Pour cela, TCP introduit un système de numéros de séquence : chaque paquet est numéroté, et les machines surveillent les numéros. Elles sont ainsi capables de détecter un paquet manquant ou dupliqué.

La surveillance des numéros de séquence demande l'allocation de ressources par le système, qui sont mobilisées pendant tout le temps que dure la communication. TCP met donc naturellement en place des communications en mode connecté. TCP introduit aussi des numéros de ports, sur 16 bits. Chaque machine dispose ainsi de 65536 ports TCP, permettant de multiplexer les communications entre processus. Chaque connexion TCP/IP est donc caractérisée par le quadruplet (adresse IP source, port TCP source, adresse IP destination, port TCP destination).

Un port TCP peut être dans différents états : en écoute, ou ouvert (quand un processus attend que l'on s'y connecte) ; en cours d'utilisation (quand un processus l'exploite pour une connexion TCP) ; fermé (quand il est inutilisé). Sur un système Unix ou Windows, on peut lire la liste des ports TCP utilisés ou en écoute avec la commande **`netstat -na`**.

PLUS LOIN **Autres états de TCP**

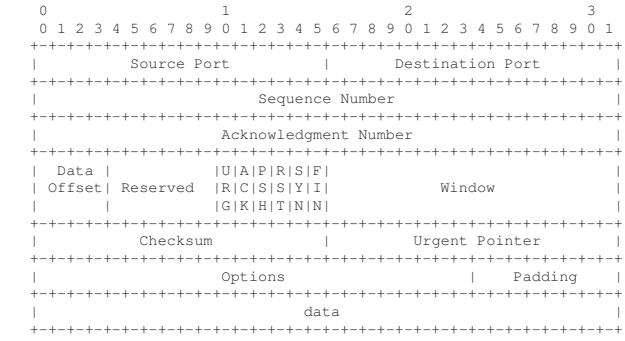
Il existe d'autres états pour les ports TCP, transitoires : ils correspondent aux ouvertures et aux fermetures de connexions. Le lecteur curieux découvrira le diagramme d'états de TCP dans la RFC 793.

EN PRATIQUE **Exemple de sortie de `netstat -na`.**

Cet exemple a deux ports en écoute (*LISTEN*) et un port utilisé par une connexion TCP (*ESTABLISHED*).

```
$ netstat -na
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        State
tcp        0      0  10.0.4.23.22           10.0.12.16.49211       ESTABLISHED
tcp        0      0  *.6000                 *.*                    LISTEN
tcp        0      0  *.22                   *.*                    LISTEN
udp        0      0  *.68
```

Voici l'en-tête TCP tel que défini par la RFC 793. Sans options, il est long de 20 octets.



TCP étant un protocole assez complexe, nous nous limiterons aux grandes lignes indispensables pour comprendre la mise en place d'un pare-feu. Dans l'en-tête

TCP, on trouve notamment les ports source et destination évoqués plus haut, le numéro de séquence, et un numéro de confirmation de séquence, qui permet d'indiquer qu'on a bien reçu un paquet.

TCP dispose aussi de drapeaux, dont les plus importants pour le concepteur de filtre sont ACK, SYN, et RST. SYN est utilisé sans ACK dans les paquets demandant l'initiation d'une connexion. Cette particularité permet de s'intéresser spécifiquement aux débuts des connexions. RST est utilisé pour terminer une connexion. Il est par exemple généré lors de tentatives de connexion à un port TCP fermé.

TCP met aussi en place des mécanismes de contrôle de flux et d'envoi de données hors bande. Le lecteur curieux se référera à un ouvrage spécialisé sur les réseaux, ou, s'il est courageux, directement à la RFC 793.

EN PRATIQUE **Perdu dans les en-têtes ?**

Faisons le point sur les en-têtes Ethernet, IP, ou TCP, avec l'agencement des en-têtes dans une trame de niveau 2 :

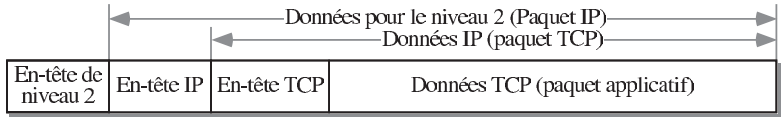


Figure 9–10 Paquet TCP vu au niveau 2

PLUS LOIN **Les drapeaux TCP**

Les curieux apprécieront la liste des drapeaux TCP et leur usage exact, même si cela sert rarement.

URG	*URGent* : indique que le paquet est à traiter de façon prioritaire.
PSH	*PuSH* : exige de TCP qu'il transmette immédiatement les données reçues au protocole de niveau supérieur.
ACK	*ACKnowledge* : ce paquet TCP accuse réception d'un paquet TCP émis dans l'autre direction. L'en-tête contient le numéro de séquence de ce dernier.
RST	*ReSeT* : demande la ré-initialisation de la connexion. En réponse à une demande de connexion, cela signifie un refus de connexion. C'est ce qui se produit quand on tente de se connecter à un port TCP non ouvert.
SYN	*SYNchronisation* : indique que l'on souhaite synchroniser les numéros de séquence.
FIN	*FINal* : ce drapeau indique une demande de terminaison de la connexion.

`tcpdump` indique les drapeaux TCP par une simple lettre. Ci-dessous, un exemple pour une demande de connexion refusée : le premier paquet porte un drapeau *Syn* (noté ici `S`), le second un drapeau *Rst* (noté `R`).

```
20:27:58.434824 violette.local.65533 > pimousse.local.23: S 4123352350:412335235
0(0) win 16384 <mss 1460,nop,wscale 0,nop,nop,timestamp 0 0> (DF)
20:27:58.434895 pimousse.local.23 > violette.local.65533: R 0:0(0) ack 412335235
1 win 0
```

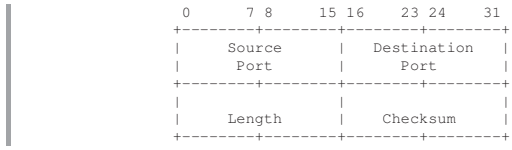
Si la fiabilité ne compte pas : UDP

TCP permet de communiquer de façon fiable ; il est extrêmement pratique car il permet aux protocoles applicatifs de ne pas se soucier des problèmes engendrés par les couches basses du réseau. On peut voir une connexion TCP comme une paire de tuyaux : tout ce qui est émis d'un côté finit par arriver de l'autre côté.

Mais TCP coûte cher. Établir et maintenir une connexion consomme des ressources systèmes, pour un bénéfice parfois nul. Pour acheminer par exemple un flux vidéo en temps réel, on se moque de la capacité de TCP à réclamer les paquets perdus. Si un paquet se perd, tant pis, on passe à la suite. Une image qui saute est préférable à un écran figé le temps de recevoir les paquets perdus : *the show must go on*.

Autre inconvénient de TCP : il ne fait que des connexions point à point. Il est incapable de transmettre vers plusieurs machines à la fois, comme dans le cadre d'une émission *multicast*.

UDP comble ces lacunes. Ce protocole se contente d'ajouter des ports UDP, sur 16 bits. Toute machine dispose donc, en plus des ports TCP, de 65536 ports UDP. Voici l'en-tête UDP tel que défini par la RFC 768.



UDP fonctionne en mode datagramme, comme des lettres envoyés par la poste. Il n'y a pas de connexion ; on envoie les données sans concertation préalable. Si elles aboutissent à un port fermé, on reçoit un message ICMP de type 3 ; si elles sont perdues en route, on ne reçoit rien ; si elles arrivent à un processus attaché au port UDP sur la machine de destination, on peut recevoir une réponse, si ce dernier décide d'en faire une.

UDP est employé dans les transactions courtes mais fréquentes, pour des raisons d'économies : les requêtes DNS, l'envoi d'informations de journalisation à travers le réseau... Il est aussi largement utilisé dans le *multicast* et dans la transmission de données en temps réel. C'est aux protocoles applicatifs de mettre en place une fiabilisation de la connexion s'ils le souhaitent.

D'autres protocoles de transport

Dans le monde IP, les autres protocoles de transport sont rares. Le fichier /etc/protocols des systèmes Unix donne la liste des numéros de protocoles encapsulables dans IP. Ce fichier est volumineux, mais renferme en pratique très peu de protocoles réellement utilisés. On y retrouve ICMP ou IGMP, protocoles de contrôle, ainsi que diverses formes d'encapsulation de protocoles de niveau 2 ou 3 dans IP.

PLUS LOIN **Numéros de protocoles**

L'en-tête IP contient un champ *Protocol*, indiquant le protocole encapsulé. Le fichier /etc/protocols donne la liste des valeurs enregistrées auprès de l'IANA, chargée de centraliser les numéros pour éviter les conflits.

Dans les autres piles réseau, on trouve d'autres protocoles de transport : NBP, ATP, RTMP, et AEP dans la pile AppleTalk ; SPX dans la pile IPX, ISO-TP dans la pile OSI...

Protocoles applicatifs

Les protocoles applicatifs, directement utilisés par les applications, sont encapsulés dans TCP ou UDP. Ils sont extrêmement nombreux, et proposent une multitude de services. Parmi les classiques, citons Telnet, HTTP, FTP, DNS, SSH, POP, IMAP, SMTP, NNTP, SMB, AFP, NFS, BootP, DHCP, LDAP, SNMP...

Le choix ne manque pas ! Certains de ces protocoles sont natifs du monde IP, d'autres existaient sur d'autres piles réseau avant d'être adaptés à IP. C'est le cas de SMB (*Server Message Block*, pour le partage de fichiers et d'imprimantes pour Windows) ou AFP (*AppleTalk Filing Protocol*, pour le partage de fichiers sur Macintosh).

Chacun de ces protocoles implique un client et un serveur. Le serveur attend les requêtes du client, en écoutant sur un port TCP ou UDP. Chaque service a un port par défaut. 80 est par exemple réservé à HTTP, 23 à Telnet, 22 à SSH, 548 à AFP...

On peut placer un serveur sur n'importe quel port, mais le client correspondant ne pourra s'y connecter que si on lui indique le bon port. Cela permet de contourner les pare-feu interdisant toute communication sur certains ports : moyennant d'installer un serveur sur un port autorisé (exemple : **`sshd`** sur le port 2222), on s'affranchit du filtrage du pare-feu.

Approfondissons la visite de quelques protocoles.

VOCABULAIRE **Ports bien connus**

En anglais, on parle de *well known ports* (ports bien connus) pour désigner les ports que ces protocoles utilisent par défaut.

PLUS LOIN **Les numéros de services**

Une fois de plus, c'est l'IANA qui enregistre les numéros de services et qui veille à l'absence de conflits. Sur les systèmes Unix, le fichier /etc/services donne la liste des services et de leur port par défaut, tels qu'enregistrés auprès de l'IANA.

Pour les pages Web : HTTP

HTTP (*HyperText Transfer Protocol*, ou protocole de transfert hypertexte) est un protocole assez simple. Il implémente quelques commandes, comme **`GET`** pour obtenir la page correspondante à une URL (adresse Web en http://), **`POST`** pour envoyer des données au serveur, et quelques autres moins communes.

HTTP fonctionne en mode déconnecté : on se connecte, récupère un ou plusieurs documents, puis se déconnecte jusqu'à ce que l'utilisateur clique sur un lien (pour une utilisation dans le cadre d'un navigateur Web) ; on se reconnecte alors, on demande le document correspondant puis on se déconnecte, et ainsi de suite. C'est pour combler les lacunes de ce mode de fonctionnement que les applications Web ont recours aux *cookies* : ils permettent de rétablir une notion de session dans la navigation Web.

HTTP est un protocole qui fonctionne en mode texte sur TCP ; on peut donc l'utiliser via la commande **`telnet`**, qui assure une connexion TCP en mode texte avec quelques séquences d'échappement pour gérer le terminal. Sous réserve de ne pas taper les séquences d'échappement, **`telnet`** fait un client HTTP minimal

B.A.-BA **Cookies et sessions**

Les *cookies* sont de petits paquets de données qui permettent au serveur de reconnaître le client et de maintenir ainsi une session entre les différentes connexions TCP qui lui parviennent. Ils sont également utilisés pour conserver des informations sur le client, que le serveur pourra récupérer plus tard.

L'alternative aux *cookies*, c'est d'utiliser des URL à rallonge, où le serveur placera les informations dont il a besoin.

tout à fait acceptable. Il faudra évidemment préciser le numéro de port à utiliser (80) – le port par défaut de Telnet est 23.

```
$ telnet www.netbsd.org 80
Trying 2001:4f8:4:b:290:27ff:feab:19a7...
Trying 204.152.184.116...
Connected to www.netbsd.org.
Escape character is '^]'.
GET /


<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<!-- Copyright (c) 1994-2003
        The NetBSD Foundation, Inc.  ALL RIGHTS RESERVED. -->
<link rev="made" href="mailto:www@NetBSD.ORG">
<link rel="shortcut icon" href="/favicon.ico" type="image/x-icon">

(...)
```

Pour monter un serveur HTTP, il faut se tourner vers Apache ou Zope, programmes tous deux disponibles dans la catégorie www des systèmes de paquetages de FreeBSD et NetBSD. Apache est aussi inclus dans la distribution de base d'OpenBSD. Nous aborderons sa configuration au chapitre 12.

PLUS LOIN **`netcat` pour tester les protocoles applicatifs**

`telnet` est un outil convenant la plupart du temps pour des sessions texte en TCP. Pour l'UDP, ou pour éviter les séquences d'échappement, on peut utiliser **`netcat`**, qui permet d'utiliser TCP et UDP directement depuis le shell.
`netcat` est disponible dans les systèmes de paquetages des BSD.

EN PRATIQUE **Port 80 ?**

Le port 80 utilisé ici est le port par défaut pour le protocole HTTP, tel que précisé dans le fichier /etc/services.

Transfert de fichiers par FTP

FTP (*File Transfer Protocol*) est un protocole de transfert de fichiers assez complet, qui permet de reprendre les téléchargements en cours de route, de transférer d'une machine à une autre sans passer par la machine locale, et bien d'autres choses encore. Comme HTTP, il fonctionne en mode texte sur TCP, mais il est un peu plus complexe.

En effet, FTP a besoin de plusieurs connexions TCP. L'une, établie par le client vers le serveur, sur le port 21, est la connexion de contrôle.

Quand le client demande le contenu d'un répertoire ou le transfert d'un fichier. FTP a besoin d'une deuxième connexion TCP, appelée connexion de données. Celle-ci peut être établie du serveur vers le client (FTP actif), ou du client vers le serveur (FTP passif).

Cette particularité de FTP le rend difficile à traiter dans le cadre d'un pare-feu. Nous verrons cela au chapitre au chapitre 10.

Pour monter un serveur FTP, on peut opter pour le serveur FTP intégré des BSD. On peut aussi choisir d'autres serveurs, plus riches en fonctionnalités, tels que WU-ftpd, disponible dans les paquetages des BSD.

Le courrier : POP3, IMAP4, SMTP

Ces trois protocoles sont utilisés pour le courrier électronique. Ils sont tous trois en mode texte et reposent sur TCP. SMTP (*Simple Mail Transfer Protocol*) est un protocole servant à l'envoi du courrier. POP3 (*Post Office Protocol version 3*) et IMAP4 (*Interactive Mail Access Protocol version 4*) permettent de relever le courrier. IMAP4 permet de conserver le courrier sur le serveur (ce qui n'est pas une

obligation). Les clients POP3 peuvent conserver le courrier sur le serveur, mais ils y arrivent avec plus ou moins de bonheur, POP3 n'étant originellement pas vraiment fait pour ça.

IMAP4 offre la gestion de boîtes multiples ainsi que quelques fonctions sympathiques, comme la commande **IDLE**, qui permet de garder la connexion au serveur ouverte et d'être prévenu à l'arrivée de nouveaux courriers. Ceci évite de contrôler la boîte aux lettres régulièrement : on reste connecté en permanence.

Comme pour HTTP, on peut parler SMTP, POP3, ou IMAP4 avec **telnet**. C'est d'ailleurs très pratique pour faire des tests ou pour lire son courrier en l'absence de client adéquat.

Voici un exemple de session SMTP réalisée grâce à **telnet** sur le port 25. Ce genre d'opérations est très classique lorsque l'on teste une configuration d'un MTA (serveur de courrier électronique) comme Sendmail. On y trouve 5 commandes SMTP : **HELO**, pour dire bonjour (SMTP est un protocole très poli) ; **MAIL FROM**, pour indiquer l'adresse d'expéditeur de l'enveloppe du courrier électronique ; **RCPT TO**, pour indiquer le destinataire de l'enveloppe ; **DATA**, pour saisir le contenu du courrier électronique à proprement parler – on y met fin par un point isolé sur sa ligne ; et **QUIT**, pour terminer la session.

```
$ telnet mail.example.com 25
Trying 192.0.2.15...
Connected to mail.example.com
Escape character is '^]'.
220-InterScan Version 3.6-Build_1201 $Date: 01/16/2002 22:21:0048$: Ready
220 mail.example.com ESMTP SendmailEAT 8.11.6/8.11.6; Sat, 8 Mar 2003 22:0
7:43 +0100;.
HELO mail.netbsd.org
250 mail.example.com Hello mail.netbsd.org [155.53.1.253], pleased to meet you
MAIL FROM: <manu@netbsd.org>
250 2.1.0 <manu@netbsd.org>... Sender ok
RCPT TO: <John.Doe@example.com>
250 2.1.5 <John.Doe@example.com>... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
From: Emmanuel Dreyfus <manu@netbsd.org>
To: John Doe <John.Doe@example.com>
Date: Sat, 8 Mar 2003 22:15:00 +0100
Subject: Et un test de sendmail, un!

Je ne sais jamais quoi dire d'intéressant dans les tests.

--
Emmanuel Dreyfus
manu@netbsd.org
.
250 2.0.0 h28L81T24462 Message accepted for delivery
QUIT
221 2.0.0 mail.example.com closing connexion
Connection closed by foreign host.
$
```

Dans le message lui-même, on retrouve les en-têtes de courrier électronique telles que définies par la RFC 822. On reproduit le `From:` et le `To:` de l'enveloppe dans le message, sinon il n'apparaîtront pas. Le corps du message est séparé de ses en-têtes par une ligne vide.

POP3 et IMAP4 peuvent être utilisés de la même façon. C'est très pratique pour déboucher une boîte ayant reçu un gros message qui la bloque. Voici un exemple pour POP3, impliquant les commandes **USER** et **PASS** pour l'authentification,

VOCABULAIRE **MUA, MTA, et MDA**

La messagerie aussi a son jargon. Le MUA est le *Mail User Agent*, c'est-à-dire le client de messagerie. Les exemples incluent Eudora, Mutt, Pine, MacSOUP, …

Le MTA est le *Mail Transfer Agent* (serveur de courrier électronique). Il reçoit le courrier du MUA et doit l'acheminer à bon port. Pour déposer le courrier dans une boîte aux lettres, le MTA utilise un MDA : *Mail Delivery Agent*, qui gère entre autres les problèmes de verrouillage des boîtes aux lettres.

LIST pour obtenir la liste des messages, **RETR** pour lire un message, **DELE** pour effacer un message, et **QUIT** pour terminer la session.

```
$ telnet pop.example.net 110
Trying 192.0.2.17...
Connected to pop.example.net.
Escape character is '^]'.
+OK Hello there.
USER manu
+OK Password required.
PASS tagadatsointsoin
+OK logged in.
LIST
+OK POP3 clients that break here, they violate STD53.
1 2505
2 2053
3 2451
4 686
.
RETR 4
+OK 686 octets follow.
Return-Path: <manu@netbsd.org>
Delivered-To: manu@example.net
Received: (qmail 9642702 invoked by uid 0); 8 Mar 2003 21:09:53 -0000
Received: from unknown (HELO mail.netbsd.org) ([155.53.1.253])
          (envelope-sender <manu@netbsd.org>)
          by 192.0.2.17 (pop.example.net) with SMTP
          for <manu@example.net>; 8 Mar 2003 21:09:53 -0000
Received: (qmail 16226 invoked by uid 1246); 8 Mar 2003 21:09:49 -0000
Delivered-To: manu@netbsd.org
Received: (qmail 15921 invoked from network); 8 Mar 2003 21:09:38 -0000
Received: from melancolie.example.net (HELO mail.netbsd.org) (192.0.2.189
) by mail.netbsd.org with SMTP; 8 Mar 2003 21:09:38 -0000

ceci est un test.
.
DELE 4
+OK Deleted.
QUIT
+OK Bye-bye.
Connection closed by foreign host.
$
```

Pour mettre en place un serveur de messagerie, les logiciels ne manquent pas. Citons notamment Sendmail, Postfix, Exim, et Qmail pour le MTA ; QPopper et UW-IMAP pour les serveurs POP ou IMAP. Sendmail est disponible dans la distribution de base des trois BSD, et nous parlerons de l'épreuve difficile et initiatique que représente sa configuration au chapitre 12. Postfix est également fourni dans la distribution de base de NetBSD, et tous les autres logiciels se trouvent dans les catégories mail des systèmes de paquetages.

Service de résolution de noms : DNS

Finissons par un service beaucoup plus fondamental : le DNS (*Domain Name System*). Il a la charge de traduire les adresses IP en adresses DNS (c'est-à-dire symboliques, en toutes lettres, et plus mnémotechniques, telles que www.apple.com), et réciproquement.

Une requête DNS comporte un nom et un type. Les plus utilisées sont les requêtes de type **A**, qui donnent une adresse IPv4, et **PTR**, qui donnent une adresse DNS. Le type **NS** donne les serveurs de noms d'un domaine, et le type **MX** donne les serveurs recevant le courrier électronique pour le domaine. Les commandes **nslookup**, **host**, et **dig** permettent d'interroger les DNS. Exemple d'utilisation :

PLUS LOIN **DNS et redondance**

Le DNS est un système distribué. Lorsqu'un serveur DNS n'a pas la réponse à une requête sur un nom de domaine, il interroge un serveur racine pour connaître le serveur faisant autorité pour le domaine, puis il contacte ce dernier. La redondance est assurée par la multiplicité des serveurs : on trouve une dizaine de serveurs racine, et chaque domaine compte au moins deux serveurs, installés en principe sur des sites différents.

```
$ nslookup -query=MX freebsd.org
Server:  gizmo.local
Address:  10.0.2.1

Non-authoritative answer:
freebsd.org     preference = 10, mail exchanger = mx1.freebsd.org

Authoritative answers can be found from:
freebsd.org     nameserver = ns0.freebsd.org
freebsd.org     nameserver = ns1.iafrica.com
freebsd.org     nameserver = ns1.downloadtech.com
freebsd.org     nameserver = ns2.downloadtech.com
freebsd.org     nameserver = ns2.iafrica.com
ns0.freebsd.org internet address = 216.136.204.126
ns1.iafrica.com internet address = 196.7.0.139
ns1.downloadtech.com    internet address = 170.208.14.3
ns2.downloadtech.com    internet address = 66.250.75.2
ns2.iafrica.com internet address = 196.7.142.133

$ nslookup -query=A www.netbsd.org
Server:  gizmo.local
Address:  10.0.2.1

Non-authoritative answer:
Name:    www.netbsd.org
Address:  204.152.184.116
```

Les requêtes et réponses DNS utilisent UDP, car elles sont courtes et fréquentes ; établir une connexion TCP à chaque fois serait plutôt pénalisant. La correction en cas d'échec est simple à faire : on envoie régulièrement des requêtes jusqu'à ce qu'on reçoive une réponse. De plus, le serveur de noms est souvent proche, on souffre donc rarement de problèmes de pertes de paquets. Cette règle souffre une exception : le transfert de zones entre DNS primaire et DNS secondaire se fait en TCP. Les serveurs n'ont pas de raison d'être proches – en fait on préfère plutôt qu'ils ne le soient pas pour assurer une meilleure résistance aux problèmes de réseau – donc utiliser UDP poserait des problèmes de fiabilité trop importants.

Sur une machine Unix qui n'est que client DNS, la configuration se fait via /etc/resolv.conf : on y précise le nom des serveurs DNS au format décrit dans la page **man** de ce fichier. Pour monter un serveur DNS, les choses sont assez simples : il faut utiliser **named**, présent dans la distribution de base des BSD. Les pages **man** de **named** et de named.conf sont des bons point de départ. Même en l'absence de nom de domaine, on peut monter un serveur DNS pour son réseau privé. Nous aborderons le détail du fonctionnement du DNS et la configuration de **named** au chapitre 12.

D'autres protocoles

On pourrait consacrer un livre entier aux protocoles applicatifs. Si ce chapitre vous laisse sur votre faim, il est temps de vous procurer un titre tel que *Computer Networks* d'Andrew S. Tanenbaum. (ISBN 0-13-394248-1), dont les centaines de pages sauront peut-être vous rassasier.

PLUS LOIN **IPv6 et le DNS**

La prise en charge d'IPv6 par le DNS est assez simple : on se contente d'ajouter un type **AAAA** qui donne une adresse IPv6. Le type **PTR** pour une adresse IPv6 donne une adresse DNS comme il le fait avec IPv4.

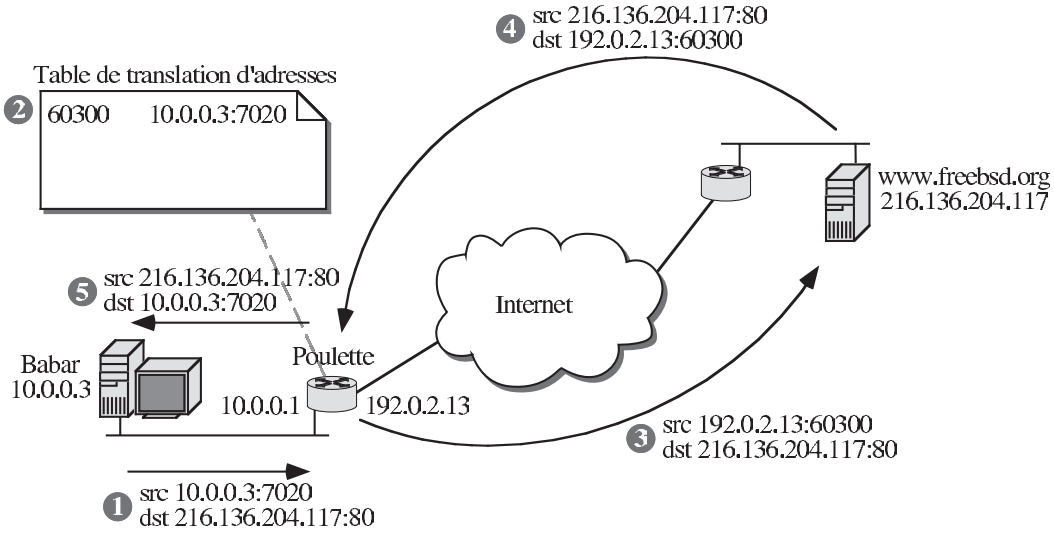
B.A.-BA **Serveurs DNS primaire et secondaire**

Chaque domaine compte au moins deux serveurs de noms faisant autorité. Pour les clients, ils sont absolument identiques, mais dans les faits, l'un dispose de la configuration maître : c'est un serveur primaire. L'autre, le serveur secondaire, n'en est qu'une réplique.
L'opération de recopie de la configuration d'un domaine DNS de serveur primaire à serveur secondaire s'appelle un transfert de zone.

CULTURE **Attribution des noms de domaines**

Les noms de domaines DNS sont gérés par l'ICANN (*The Internet Corporation for Assigned Names and Numbers*). Pour les différents gTLD (*generic Top Level Domain*, ou domaines de premier niveau génériques, tels que `.com`, `.net` ou `.org`), l'ICANN désigne des organismes chargés de la distribution des noms de domaines. Ces derniers vendent les noms de domaines à l'année, selon la règle du premier arrivé, premier servi. Quelques règles permettent toutefois de résoudre les litiges.
Dans le cas des ccTLD (*Country Code Top Level Domain* : domaines nationaux tels que `.fr` ou `.uk`), chaque pays est maître de la politique d'allocation. En France, c'est l'AFNIC (Association Française pour le Nommage Internet en Coopération) qui est chargée de la gestion des noms de domaine en `.fr`.
Les gTLD et ccTLD sont souvent désignés conjointement par le terme « TLD ».

10


4 src 216.136.204.117:80
dst 192.0.2.13:60300
Table de translation d'adresses
2 60300 10.0.0.3:7020
www.freebsd.org
216.136.204.117
5 src 216.136.204.117:80
dst 10.0.0.3:7020
Internet
Babar
10.0.0.3
Poulette
10.0.0.1
192.0.2.13
3 src 192.0.2.13:60300
dst 216.136.204.117:80
1 src 10.0.0.3:7020
dst 216.136.204.117:80

BSD dans le rôle du pare-feu

Les systèmes BSD sont assez appréciés pour construire des pare-feu (on voit souvent le terme anglais *firewall*). Ils permettent de monter une solution évolutive et robuste pour un coût minimal.

SOMMAIRE

MOTS-CLEFS

- Routage
- Pare-feu
- Filtrage et ACL
- Translation d'adresses
- VPN

Routage avec BSD

Un simple routeur dépourvu de fonction de filtrage ou de régulation de trafic n'a qu'un intérêt limité au routage entre deux types de liens différents (réseau Ethernet et liaison spécialisée, par exemple), à l'introduction d'un routage au niveau 3 dans un réseau pour en rationaliser le trafic, ou aux expériences éducatives.

Le routage est aussi la première pierre de dispositifs plus complexes, avec filtrage, translation d'adresses, redirections de ports, etc. Examinons ce qu'il est possible de mettre en place avec un système BSD.

Besoins matériels

Un routeur est une machine dédiée. Première bonne surprise : les besoins en performances sont faibles : faire passer des paquets d'une interface à l'autre ne sollicite pas le disque dur, requiert peu de mémoire et peu de puissance de calcul.

À titre d'exemple, un Macintosh IIci est équipé d'un 68030 cadencé à 25 MHz. Utilisé comme routeur sur un réseau Ethernet à 10 Mb/s entre un client et un serveur FTP, il fournira un trafic très régulier de 1,5 Mb/s. Il n'atteint pas les 10 Mb/s, mais cette expérience assure qu'un Macintosh II est assez puissant pour servir de routeur sur une liaison câblée, ADSL, voire une petite liaison spécialisée. L'auteur utilise d'ailleurs un Macintosh acheté douze euros comme routeur sur une connexion câblée.

Si la machine est chargée de filtrage à états (concept présenté plus loin) ou de translation d'adresses, le noyau devra traiter des tables listant les communications en cours. Plus le réseau sera étendu, plus les besoins en mémoire et en processeur pour stocker et parcourir ces tables seront importants.

Un exemple illustrera le propos. Sur un réseau d'un millier de postes lié à l'extérieur à 100 Mb/s, un PC sous NetBSD à 800 MHz et de 32 Mo de mémoire suffira amplement pour du filtrage à états. Le grand nombre de connexions traversant le filtre (de 5000 à 8000) demandera quelques ajustements dans la configuration du noyau, mais tout se passera bien s'il n'y a pas d'erreur grossière dans les règles de filtrage.

ALTERNATIVE **Routeurs matériels**

On trouve sur le marché des produits intégrant un pare-feu dans un petit commutateur, destinés à la protection des réseaux domestiques. Une machine sous Unix est plus difficile à mettre en œuvre, mais elle dispose d'une bien plus grande flexibilité.

Pour les réseaux d'entreprise, on trouve aussi des matériels plus robustes, uniquement pare-feu. Le pare-feu à base de PC sous Unix fournira des fonctionnalités similaires pour un coût très inférieur.

Configuration des interfaces

Router suppose d'abord de configurer les interfaces réseau. Nous avons vu au chapitre 7 que dans les cas simples (interface Ethernet et configuration manuelle), ceci se faisait avec la commande **`ifconfig`** pour la plupart des systèmes Unix. On ne reviendra donc pas sur ce point, hormis pour certaines particularités spécifiques aux pare-feu et aux routeurs.

Quelle carte est la bonne ?

Les routeurs comptent souvent plusieurs cartes Ethernet. Une des difficultés consiste à savoir quelle carte correspond à quelle interface réseau. Dans le cas général, la solution consiste à configurer une interface et à tenter un **`ping`**. S'il passe, on a configuré la bonne carte ; dans le cas contraire, on échange. C'est peu pratique, mais sur les cartes récentes une astuce permet de conclure plus rapidement.

Exemple de sortie de la commande **`ifconfig`**.

```
$ ifconfig -a
ex0: flags=8863<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        address: 00:c0:4f:b7:f3:3b
        media: Ethernet autoselect (100baseTX full-duplex)
        status: active
        inet 192.168.3.21 netmask 0xffffff00 broadcast 192.168.3.255
fxp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        address: 00:03:47:73:12:f9
        media: Ethernet autoselect (100baseTX full-duplex)
        status: active
        inet 10.54.82.50 netmask 0xffffff00 broadcast 10.54.82.255
lo0: flags=8009<UP,LOOPBACK,MULTICAST> mtu 33220
        inet 127.0.0.1 netmask 0xff000000
ppp0: flags=8010<POINTOPOINT,MULTICAST> mtu 1500
ppp1: flags=8010<POINTOPOINT,MULTICAST> mtu 1500
sl0: flags=c010<POINTOPOINT,LINK2,MULTICAST> mtu 296
sl1: flags=c010<POINTOPOINT,LINK2,MULTICAST> mtu 296
```

La ligne `status` indique si la carte est bien reliée à un autre équipement réseau : « `active` » dans l'affirmative, elle indiquera « `no carrier` » sinon. Il suffit donc de déconnecter une carte et d'exécuter à nouveau **`ifconfig -a`** pour savoir quelle carte on vient de débrancher. C'est une bonne idée d'étiqueter les cartes à l'arrière de la machine pour éviter toute hésitation ultérieure.

Des adresses, des interfaces, beaucoup de possibilités

Quand on sait à quelle interface correspond chaque carte, il est temps d'attribuer les adresses. Comme on l'a vu au chapitre 7, on affecte une adresse IPv4 à l'interface `ex0` en tapant une commande comme suit :

```
# ifconfig ex0 inet 192.168.3.21 netmask 255.255.255.0
```

Le mot-clef **`inet`** dénote une adresse IPv4 ; **`inet6`**, une adresse IPv6.

On peut aussi préciser un alias, c'est-à-dire une deuxième adresse sur la même interface. Sur NetBSD et OpenBSD :

```
# ifconfig ex0 alias 192.168.4.21 netmask 255.255.255.0
```

Après ces deux commandes, l'interface `ex0` porte deux adresses IP dans deux sous-réseaux différents. Ce type de configuration permet de router entre deux sous-réseaux qui cohabitent sur le même réseau physique. Dans cette situation, le routeur n'a besoin que d'une seule interface pour faire son travail.

EN CAS DE COUP DUR **Pas de ligne `status`**

Si **`ifconfig`** n'affiche pas de ligne `status`, cela indique tout simplement que le pilote de la carte est incapable de détecter la connexion au réseau. Soit qu'il ne sait pas le faire, soit car la carte ne fournit pas cette information (c'est le cas sur les cartes les plus anciennes).
Pour identifier la carte, il faut alors avoir recours à **`ping`**.

PLUS LOIN **Serveurs virtuels**

Il est possible de configurer plusieurs adresses du même sous-réseau sur une seule interface. Cette configuration sert à utiliser plusieurs services sur une même machine et donner l'illusion que chacun est hébergé sur une machine différente. On accumule alors les adresses sur la même interface, chacune ne servant que pour un seul service.
Nous verrons un exemple d'utilisation de cette configuration avec les serveurs Web virtuels d'Apache, au chapitre 12.

SUR LES AUTRES UNIX **FreeBSD et les alias**

Sous FreeBSD, la syntaxe est légèrement différente : **`ifconfig ex0 inet 192.168.4.21 netmask 255.255.255.0 alias`**.
Attention, si vous configurez sur une interface plusieurs adresses dans le même sous-réseau, alors FreeBSD exigera un masque en `255.255.255.255` pour les alias.

PLUS LOIN **Bridges**

Tout comme on peut multiplier les adresses sur une interface, on peut utiliser la même adresse sur plusieurs interfaces. C'est ce qui se passe lors de la configuration d'un *bridge* (pont), cas traité plus loin.

Le routeur ne route pas !

Configurer les interfaces ne suffit pas : par défaut, le système ne route pas, et les paquets parvenant sur une de ses interfaces ne pourront pas le traverser. Sous BSD, on active le routage avec la commande **sysctl**.

sysctl permet de manipuler des variables qui régissent le fonctionnement du noyau. **sysctl -a** affiche une longue liste de toutes les variables :

```
$ sysctl -a
kern.osrelease = 1.6M
kern.osrevision = 106130000
kern.version = NetBSD 1.6M (DARWIN) #12: Fri Jan 31 23:42:16 UTC 2003
    manu@violette:/mnt/src/sys/arch/macppc/compile/DARWIN

kern.maxvnodes = 2097
kern.maxproc = 532
kern.maxfiles = 1772
(...)
```

Seules nous intéressent les variables régissant le comportement du routage IPv4 (voire IPv6) :

```
(...)
net.inet.ip.forwarding = 0
net.inet6.ip6.forwarding = 0
(...)
```

Le zéro indique que le routage est désactivé. Pour l'activer, on emploiera la commande **sysctl** avec l'option **-w** :

```
# sysctl -w net.inet.ip.forwarding=1
net.inet.ip.forwarding: 0 -> 1
#
```

ALTERNATIVE **FreeBSD et sysctl**

Sous FreeBSD, l'option **-w** de **sysctl** est facultative.

SÉCURITÉ **Désactiver le routage**

Pour désactiver le routage, **sysctl** est encore l'outil à utiliser. La plupart des services s'arrêtent en tuant le processus qui les assure, mais le routage n'implique aucun processus : c'est le noyau lui-même qui fait le travail. Même un passage en mode mono-utilisateur n'arrêtera pas le routage ; il faut le demander explicitement.

ALTERNATIVE **Activation du routage sur FreeBSD**

Sur FreeBSD, on peut aussi activer le routage au démarrage en indiquant **gateway_enable="YES"** dans le fichier /etc/rc.conf.

Reconfiguration au démarrage

On a vu au chapitre 7 comment configurer le réseau au démarrage de la machine. Pour une configuration en routeur, il suffit d'ajouter l'activation du routage, en faisant automatiquement exécuter la commande **sysctl -w net.inet.ip.forwarding=1**. On utilise pour cela le fichier /etc/sysctl.conf, renfermant des couples **variable=valeur** à initialiser au démarrage du système. Exemple typique :

```
net.inet.ip.forwarding=1
```

Routage statique ou dynamique

Table de routage

Le routeur sait comment acheminer les paquets destinés aux sous-réseaux où il dispose d'une interface. Pour aller plus loin, il faut définir une table de routage.

Elle se situe dans le noyau, et chacune de ses entrées associe un sous-réseau et l'adresse d'une passerelle qui y mène, passerelle qui sera directement accessible. Exemple de table de routage :

Destination	Passerelle
0.0.0.0/0	192.168.3.1
10.10.0.0/16	192.168.3.2
192.168.3.0/24	ex0
127.0.0.0/8	lo0

Cette table comprend le réseau local, `127.0.0.0/8`, auquel on accède par l'interface locale (`lo0`). La machine est dotée d'une carte réseau avec une adresse dans le sous-réseau `192.168.3.0/24`, accessible directement par l'interface `ex0` (qui a sûrement une adresse dans ce sous-réseau).

On trouve enfin deux entrées de routage : l'une pour le réseau `10.10.0.0/16`, dont on enverra tout le trafic à la machine `192.168.3.2`, l'autre pour l'ensemble de l'espace d'adresses IPv4, soit `0.0.0.0/0`. Cette adresse indique la route par défaut pour tous les paquets qui ne sont pas concernés par les autres règles : tout envoyer à la machine `192.168.3.1`.

La commande `route`

La commande **`route`** permet de manipuler la table de routage. Sa syntaxe différant légèrement entre les BSD, la lecture de sa page **`man`** s'impose.

À titre d'exemple, en voici quelques utilisations :

```
# route show
Routing tables

Internet:
Destination       Gateway            Flags
default           10.0.12.1          UG
10.0.12.0         link#1             U
gizmo.local       00:00:c5:41:30:0d  UH
trotsky.local     00:05:02:96:9b:52  UH
loopback          127.0.0.1          UGR
localhost         127.0.0.1          UH
# route add 192.168.1.0 -netmask 255.255.255.0 10.0.0.1
add net 192.168.1.0: gateway 10.0.0.1
# route delete -net 192.168.1.0 -netmask 255.255.255.0
delete net 192.168.1.0
#
```

PIÈGE À C… **Se couper la route**
Attention lors de la manipulation de la commande **`route`** à distance. Une mauvaise manipulation pourra couper la route qui mène à la machine. Dans le pire des cas, il faudra ensuite se déplacer physiquement sur la console concernée pour reprendre la main.

Ces commandes sont assez claires : la première affiche la table, la deuxième ajoute le réseau `192.168.1.0/24`, accessible par la passerelle `10.0.0.1`, et la troisième ligne supprime immédiatement cette nouvelle entrée.

L'option **`show`** est spécifique à NetBSD et OpenBSD. La commande standard sous Unix pour afficher la liste des routes du système est **`netstat -r`**.

La commande **`route`** tente par défaut de résoudre les adresses. Si le DNS est inaccessible (ce qui arrive parfois lorsque l'on manipule les routes), il faudra patienter à chaque échec de requête DNS. Deux solutions : inscrire dans le fichier /etc/hosts toutes les adresses utilisées dans les routes, ou utiliser l'option **`-n`** de **`route`**, qui indique de ne pas résoudre les noms.

RAPPEL **L'option `-n` et le DNS**
Si le DNS n'est pas accessible, **`netstat -r`** et **`route show`** seront assez pénibles à utiliser. L'option **`-n`** leur indiquera de ne pas tenter de résoudre les adresses IP en adresses DNS.

Reconfiguration des routes statiques au démarrage

Voilà encore un point où chaque système d'exploitation procède à sa manière ; cette reconfiguration n'est même pas toujours prévue par les scripts de démarrage. Dans tous les cas, on peut faire simple et placer les commandes adéquates dans le fichier /etc/rc.local.

Protocoles de routage dynamique

Les entrées ajoutées à la main dans les tables de routage sont dites statiques. C'est clairement la voie à suivre pour gérer un réseau comptant peu de sous-réseaux et de routeurs.

Mais cette méthode est inappropriée dans le cas d'un réseau de nombreux routeurs, pas toujours maintenus par les mêmes personnes. Pour un campus universitaire en comptant des dizaines, toute mise en place d'un nouveau sous-réseau et de son bloc d'adresses imposerait d'ajouter une route statique sur chaque routeur... démarche peu efficace.

La solution, ce sont les protocoles de routage dynamique. Ils permettent aux routeurs de s'échanger des routes, donc de répercuter automatiquement tout changement dans les tables de routage en y ajoutant des entrées alors qualifiées de dynamiques.

Il existe de nombreux protocoles de routage dynamique. Pour le routage intérieur : RIP, IS-IS, OSPF ; pour le routage extérieur entre systèmes autonomes (AS ou *Autonomous Systems*) : EGP, BGP... Chacun d'entre eux est implémenté dans un *daemon*, comme **routed** pour RIP, **gated** pour BGP, et **zebra** pour BGP ou OSPF. Ce *daemon* échange des paquets avec les autres *daemons* des routeurs du réseau, et met à jour les tables de routage de façon adéquate.

La mise en œuvre de protocoles de routage dynamique dans un grand réseau dépasse largement le cadre de cet ouvrage. La lecture de pages **man** comme celle de **routed**, mettra le pied à l'étrier, mais il faudra probablement consulter un livre spécialisé pour bien maîtriser la question.

PLUS LOIN **À propos de BGP**

BGP (*Border Gateway Protocol*) est le protocole utilisé pour la propagation des règles de routages sur Internet.
Internet est une interconnexion de réseaux sans politique de routage par défaut, appelés « systèmes autonomes », qui comptent chacun au moins deux liens vers l'extérieur. Lorsque les routeurs en bordure d'un système autonome doivent envoyer un paquet à l'extérieur, ils consultent des tables de routage au lieu de passer le paquet à une passerelle par défaut. Le paquet sortant sera envoyé sur le lien qui correspond à la route la plus courte ou la moins onéreuse.
Le protocole BGP permet aux routeurs de bordure des systèmes autonomes de connaître toutes les routes sur Internet, et d'avertir le reste du monde de toute nouvelle route chez eux. Ces machines devant manipuler des tables de routages de plusieurs dizaines de milliers d'entrées, elles ont de gros besoins en mémoire et en puissance de calcul. On confie souvent ce travail à des routeurs spécialisés (comme ceux de Cisco), bâtis pour cette tâche.

ATTENTION **Chargement des routes statiques**

/etc/rc.local n'est pas toujours le bon endroit pour charger les routes statiques, par exemple si elles doivent être utilisées par un *daemon* invoqué avant son exécution.
On peut alors ajouter un script dans la séquence d'initialisation pour configurer les routes avant l'invocation de ce *daemon*, ou redémarrer celui-ci dans /etc/rc.local après le chargement de toutes les routes. La meilleure solution est d'utiliser les éventuels moyens mis à disposition par le système d'exploitation. Sous NetBSD, on peut par exemple créer un script /etc/netstart.local, qui sera exécuté à la configuration du réseau, avant le démarrage des *daemons*. Exemple :

```
#!/bin/sh
route add 192.168.5.0 -netmask 255.255.255.0 192.168.3.1
route add 172.16.0.0 -netmask 255.255.0.0 192.168.3.2
```

Sous FreeBSD, on peut ajouter des routes statiques par le biais du fichier /etc/rc.conf :

```
static_routes="rd sales"
route_rd="192.168.5.0 -netmask 255.255.255.0 192.168.3.1"
route_sales="172.16.0.0 -netmask 255.255.0.0 192.168.3.2"
```

Bridges

Les *bridges* (ponts) sont des dispositifs relayant les paquets au niveau 2 entre deux réseaux. On les utilise parfois dans les dispositifs de filtrage pour monter des pare-feu transparents.

Pourquoi monter un *bridge* ?

Sur un *bridge*, la machine transmet des paquets d'une interface à l'autre sans faire de routage : elle se comporte de façon transparente et recopie simplement le trafic. Les deux interfaces n'ont même pas besoin d'avoir des adresses IP.

Les *bridges* essaient de limiter le trafic et de ne passer que les paquets nécessaires : ceux dont la machine destination est bien de l'autre côté. En ce sens, ils fonctionnent comme des commutateurs : ils travaillent d'abord en phase d'apprentissage, en laissant tout passer. Quand ils connaissent l'emplacement d'une machine, ils ne transmettent le trafic que si nécessaire : quand deux machines situées du même côté d'un *bridge* dialoguent entre elles, celui-ci ne transmettra rien.

Les *bridges* ont plusieurs applications, évoquées au chapitre précédent. Ils permettent de réduire le trafic réseau, comme un commutateur. Ils permettent aussi de relier entre eux deux réseaux de types différents, sans faire de routage. On peut encore les utiliser comme filtres transparents. Un routeur filtrant est visible, puisqu'il travaille au niveau 3 ; **`traceroute`**, peut déceler sa présence. Un *bridge*, qui ne travaille qu'au niveau 2, n'est pas détectable facilement. Cet avantage peut se muer en inconvénient : il peut être plus difficile de diagnostiquer le blocage de paquets par le filtre.

Autre avantage des filtres transparents : il n'est pas nécessaire de réorganiser le réseau pour les mettre en place. On les positionnera souvent entre une passerelle par défaut permettant de sortir du réseau et le reste du réseau. Dans le cas d'un routeur filtrant, il faut créer un sous-réseau entre la passerelle et le filtre, ce qui impose une petite réorganisation.

Dernière application du *bridge* : c'est un formidable outil de diagnostic dans un réseau commuté. En effet, il y est impossible d'écouter simplement les communications entre deux machines depuis une troisième. Un portable doté de deux interfaces réseau employé comme *bridge* permet de s'intercaler entre une machine et son commutateur, et ainsi d'observer tout le trafic qui lui est destiné.

PLUS LOIN **Espionnage en réseau commuté**

L'examen du trafic qui passe sur le réseau n'est hélas pas l'exclusivité de l'administrateur système qui essaie de diagnostiquer un problème. Les pirates emploient aussi ces techniques.

Il existe d'autres moyens de faire de l'espionnage en réseau Ethernet commuté. Le programme **`ettercap`** permet par exemple d'émettre des paquets ARP pour détourner tout le trafic destiné à une adresse IP. Les paquets, une fois examinés, sont ensuite renvoyés à leur destinataire légitime.

Contrairement à l'usage d'un *bridge*, cette technique n'impose pas de s'intercaler à un endroit donné ; un pirate pourra la mettre en œuvre depuis n'importe quelle machine du réseau dont il aura pris le contrôle à distance. Heureusement, la discrétion n'est pas au rendez-vous, et la surveillance du trafic ARP permettra de détecter cette activité suspecte (en utilisant par exemple un programme comme **`arpwatch`**).

Mise en œuvre

Une fois de plus, la mise en œuvre dépend du système employé. Sous FreeBSD, on utilise **`sysctl`**. Exemple issu de la page **`man`** de **`bridge`** :

```
# sysctl net.link.ether.bridge_cfg=dc0:3,dc1:3,fxp0:4,fxp1:4
```

On y crée deux *bridges*, « 3 » entre les interfaces `dc0` et `dc1`, et « 4 » entre les interfaces `fxp0` et `fxp1`.

ATTENTION **Attribution des noms d'interfaces**

On suppose ici l'existence d'interfaces appelées `dc0` et `dc1`, mais ce nom n'est actuellement pas utilisé sous NetBSD et OpenBSD.

RAPPEL **Interfaces clonables**

`bridge0` est une interface « clonable » : **`ifconfig -a`** ne la mentionne pas. On peut créer de telles interface à volonté en utilisant le mot-clef **`create`** dans **`ifconfig`**.
La commande **`ifconfig -C`** donne la liste des interfaces clonables disponibles.

Sous NetBSD et OpenBSD, on utilise deux commandes : **`ifconfig`** crée une pseudo-interface *bridge* et **`brconfig`** permet de la configurer. La configuration décrite ci-dessus s'écrira ainsi :

```
# ifconfig bridge0 create
# brconfig bridge0 add dc0 add dc1 up
# ifconfig bridge1 create
# brconfig bridge1 add fxp0 add fxp1 up
```

Le *bridge* constitué de `dc0` et `dc1` s'appelle `bridge0` ; le *bridge* constitué de `fxp0` et `fxp1` s'appelle `bridge1`. La syntaxe, plus verbeuse que celle de FreeBSD, a l'avantage de créer une interface réseau, que l'on peut manipuler comme n'importe quelle autre.

On détruit un *bridge* comme suit :

```
# ifconfig bridge0 destroy
```

Filtrage du trafic

Le filtrage est une occupation très courante pour les systèmes BSD. Dans cette section, nous introduirons les méthodes de filtrage IP. Attention : aucun des BSD n'est correctement équipé pour filtrer des protocoles comme AppleTalk ou IPX.

Trois systèmes BSD, trois filtres différents

À l'origine, les trois BSD partageaient un même filtre : IPFilter, qu'on trouve aussi sur d'autres systèmes d'exploitation, comme Solaris, HP-UX, ou Linux 2.0. FreeBSD a un deuxième filtre, IPFW, qui peut remplacer ou compléter IPFilter. Récemment, OpenBSD a abandonné IPFilter pour développer PacketFilter. Désormais les trois BSD utilisent donc trois filtres différents. IPFilter étant présent sur NetBSD et FreeBSD, on l'utilisera pour illustrer la mise en œuvre des *Access Lists* (ACL) sur un pare-feu.

SUR LES AUTRES UNIX **Filtrage de paquets**

On trouve IPFilter sur de nombreux Unix : Solaris, HP-UX ; on peut même l'installer sur un vieux Linux 2.0…
Sous GNU/Linux les deux logiciels de filtrage les plus répandus sont IPChains et IPTables, interfaces utilisateur vers le même filtre dans le noyau.

PacketFilter diffère de IPFilter en plusieurs points, mais tous deux restent assez semblables dans leur philosophie, et le passage de l'un à l'autre est en majeure partie une affaire de syntaxe des fichiers de configuration. La dernière partie de ce chapitre aborde les différences entre ces deux filtres.

Construire son ACL

Principes généraux

Le filtre est gouverné par une ACL (*Access List*, ou liste d'accès), liste de règles indiquant ce qui doit traverser le filtre et ce qui doit être bloqué.

L'ACL d'IPFilter est stockée dans un fichier texte : /etc/ipf.conf pour NetBSD, et /etc/ipf.rules pour FreeBSD. Pour modifier les règles, il suffit de modifier le fichier et d'invoquer la commande indiquant de détruire les règles existantes et d'y charger les nouvelles : **`ipf -Fa -f /etc/ipf.conf`**. La page **`man ipf`**

fournira les détails. Pour la première utilisation de IPFilter, il faut initialiser le filtre avec la commande **`ipf -E`**.

Sur NetBSD, sous réserve d'écrire **`ipfilter=YES`** dans /etc/rc.conf, les scripts d'initialisation s'occuperont de l'initialisation du filtre et du chargement des règles de /etc/ipf.conf. On obtient le même résultat sous FreeBSD en indiquant **`ipfilter_enable="YES"`** dans /etc/rc.conf.

Quand un paquet se présente, le filtre examine toutes les règles et cherche celles qui s'appliquent. Si une seule convient, les choses sont simples. Si plusieurs règles correspondent, tout dépend du filtre : certains logiciels utilisent la première règle trouvée, d'autres, comme IPFilter, la dernière.

Si aucune règle ne concerne le paquet, tout dépend encore une fois du comportement par défaut du filtre : certains laissent passer, d'autres bloquent, d'autres, comme IPFilter, sont configurables.

Pour éviter d'écrire une ACL dont le comportement dépend du réglage du comportement par défaut, il vaut mieux commencer par une règle qui bloque tout ou qui laisse tout passer, selon l'effet cherché. Bien souvent, pour être sûr de ne pas laisser passer de paquets par erreur, on commence par tout interdire puis autorise ce qui doit l'être. Avec IPFilter, cela s'écrit :

```
block in from any to any
```

On peut aussi choisir de bloquer les paquets sortants par défaut en ajoutant la règle **`block out from any to any`**, mais cela apporte peu en matière de sécurité : à l'exception des paquets émis par la machine filtrante elle-même, les paquets sortants sont forcément d'abord entrés ; autant les bloquer à l'entrée.

On peut désormais ajouter quelques règles permissives. Laissons par exemple passer tout le trafic local, sur l'interface `lo0` :

```
block in from any to any
pass in on lo0 from any to any
```

Dans cette ACL, un paquet passant sur `lo0` est bloqué par la première règle mais autorisé à passer par la deuxième. La dernière règle l'emportant, ce paquet sera autorisé à passer. Tout autre paquet sera bloqué par la première règle ; non autorisé par les règles suivantes, il sera bloqué par le filtre.

Pour autoriser ensuite le trafic d'origine ou de destination le sous-réseau `10.10.0.0/16`, on écrira :

```
block in from any to any
pass in on lo0 from any to any
pass in from 10.10.0.0/16 to any
pass in from any to 10.10.0.0/16
```

Et ainsi de suite... Le principe est assez simple. Étudions maintenant les possibilités d'expression des règles d'IPFilter.

SÉCURITÉ **Modification des règles**

`ipf -Fa -f /etc/ipf.conf` met à jour les règles, mais pendant une brève fenêtre temporelle le filtre sera vide. IPFilter propose un mécanisme pour éviter cela : les règles de filtrage actives peuvent être doublées d'un jeu de règles de filtrage inactives, qu'on chargera en tapant **`ipf -I -Fa -f /etc/ipf.conf`**. On basculera ensuite d'un jeu à l'autre avec **`ipf -s`**.
Plus simplement, on peut utiliser le script de démarrage d'IPFilter, qui reprend ces commandes : **`/etc/rc.d/ipfilter reload`**

B.A.-BA **L'interface locale**

`lo0` est l'interface locale, d'adresse `127.0.0.1`, et qui permet à la machine de se connecter à elle-même sans passer par le réseau.

Petit tour d'horizon des possibilités d'IPFilter

Filtrage sur un protocole donné :

```
pass in on ex0 proto tcp from any to any
```

Sur un port TCP ou UDP :

```
block in on fxp0 proto tcp from 10.0.0.0/24 to any port = 23
block in on fxp0 proto tcp from 10.0.0.0/24 port <= 1024 to any
pass in on ex0 proto udp from 10.0.0.0/24 port >= 1024 to any port >= 1024
```

On peut indiquer qu'au lieu de bloquer le paquet de façon silencieuse, le filtre doit envoyer un paquet ICMP mentionnant que le port, la machine, ou le réseau est inaccessible. Pour les paquets TCP, on peut demander le renvoi d'un paquet TCP RST indiquant une connexion refusée.

```
block in return-icmp(net-unr) from any to any
block in return-rst proto tcp from any to any
```

On sait très rapidement de certains paquets qu'ils doivent être bloqués. Il faut donc cesser le parcours de l'ACL, pour un traitement plus rapide et pour éviter de laisser passer ces paquets suite à une erreur plus loin dans le jeu de règles.

Le mot-clef `quick` stoppe immédiatement le parcours de l'ACL :

```
pass in quick on lo0 from any to any
```

Ceci n'est qu'une fraction des possibilités offertes par IPFilter. Le but est de fournir au lecteur assez de vocabulaire pour lui permettre de construire des filtres.

On trouve beaucoup d'autres fonctionnalités très utiles, comme la possibilité de construire des ensembles de règles, dont l'examen est subordonné au passage dans une règle. Ceci se fait avec les mots-clef `head` et `group`. Le lecteur curieux se reportera à la documentation pour en savoir plus.

PLUS LOIN **Documentation sur IPFilter**

Ce chapitre ne présente évidemment pas IPFilter de manière exhaustive. On trouve une bonne documentation dans les pages **man**, mais aussi sur Internet. Le site Web http://www.obfuscation.org/ipf est une mine d'informations, comprenant notamment le très complet IPFilter « *HOW-TO* » : http://www.obfuscation.org/ipf/ipf-howto.html.

Une ACL simple mais complète

Voici à titre d'exemple une ACL complète. La machine a deux cartes, `ex0` (`192.168.10.1/24`) et `fxp0` (`192.168.0.1/30`). `ex0` donne sur un réseau interne que l'on souhaite protéger, et `fxp0` donne sur l'extérieur.

Nous souhaitons autoriser les trafics TCP, UDP, et ICMP, et empêcher les accès depuis l'extérieur aux services TCP ou UDP sur les ports inférieurs à 1024. Il faut en outre se protéger contre l'*IP spoofing*, et s'assurer qu'aucun paquet ne peut venir de l'extérieur en prétendant être issu du réseau interne.

```
block in return-icmp from any to any
block in return-rst proto tcp from any to any (1)

block in log quick from any to any with short (2)

block in return-rst quick proto tcp from any to any with ipopts
block in return-icmp quick from any to any with ipopts (3)

pass in quick on lo0 from any to any (4)

block in log quick from any to 127.0.0.0/8
```

```
block in log quick from 127.0.0.0/8 to any
block in log quick from 192.168.10.1/32 to any
block in log quick from 192.168.0.1/32 to any ❺

block in log quick on ex0 from !192.168.10.0/24 to any
block in log quick on ex0 from any to 192.168.10.0/24 ❻

block in log quick on fxp0 from 192.168.10.0/24 to any ❼

pass in proto tcp from any to any
pass in proto udp from any to any
pass in proto icmp from any to any ❽

block in return-rst on fxp0 proto tcp from any to any port < 1024
block in return-icmp on fxp0 proto udp from any to any port < 1024 ❾
```

❶ On commence par tout bloquer avec notification. Dans le cas général, on notifie par paquet ICMP, mais pour TCP on envoie un paquet RST. La règle pour TCP, plus spécifique, sera placée ensuite : c'est en effet la dernière règle qui l'emporte.

❷ On bloque les paquets trop courts pour être honnêtes, et on les consigne dans les journaux.

❸ Certaines options IP posent des problèmes de sécurité ; on les bloque donc par défaut. La règle pour TCP précède ici la règle générale : grâce au `quick`, elle va l'emporter bien qu'elle apparaisse avant.

❹ Tout le trafic sur `lo0` est autorisé.

❺ Grâce au `quick` pour le trafic sur `lo0`, les règles suivantes ne sont pas évaluées pour cette interface. On évacue ainsi un certain nombre de monstruosités, comme les paquets venant du réseau avec pour adresse source `127.0.0.1`.

❻ Sur `ex0` (interface interne), on bloque tout ce qui tente de sortir avec pour adresse source une adresse externe ou pour adresse de destination une adresse interne.

❼ Sur `fxp0`, on bloque de même ce qui prétend venir d'une adresse du réseau interne.

On a désormais évité l'*IP spoofing* contre le réseau interne ou le routeur lui-même.

❽ On autorise TCP, UDP et ICMP...

❾ ...sauf ce qui entre en TCP et UDP sur les ports inférieurs à 1024.

SÉCURITÉ **IP spoofing**

L'*IP spoofing* consiste à émettre des paquets avec une adresse IP source fausse. Cette technique sert dans toute une panoplie d'agressions informatiques. On peut par exemple émettre des **`ping`** en *broadcast* sur tout un réseau en indiquant comme adresse source l'adresse IP de quelqu'un d'autre. Pour chaque paquet ainsi envoyé, la victime recevra plusieurs centaines de réponses, ce qui engorgera son réseau. C'est l'attaque dite du « *smurf* ».

Filtres à états

Lors de la composition d'un filtre, il faut se rappeler que le trafic est bidirectionnel. Si un paquet est autorisé dans un sens, une règle doit laisser passer sa réponse. On obtient ainsi des situations où il faut être bien plus permissif qu'on ne le souhaiterait.

Par exemple, pour communiquer en UDP avec l'extérieur, il faut laisser sortir les paquets UDP, mais aussi laisser rentrer leurs réponses, c'est-à-dire tout le trafic UDP. Ceci peut s'avérer peu souhaitable car un tel trafic pourrait être utilisé comme sonde, ou comme attaque contre un service UDP vulnérable dans le réseau interne.

Les filtres à états traitent cette catégorie de problèmes. Dans un filtre à états, la machine consigne tout paquet UDP transmis dans une « table des états ». Elle permettra d'identifier en tant que tel tout paquet ICMP ou UDP reçu en réponse ; la machine le laissera alors passer. Mais un paquet UDP sans rapport avec aucun paquet émis sera rejeté.

IPFilter permet de préciser que l'on souhaite accepter les paquets reçus en réponse d'un paquet donné grâce à l'option **`keep state`** :

```
pass in on ex0 proto udp from any to any keep state
```

Cette règle laisse passer les paquets UDP sortants sur `ex0` et accepte sur `fxp0` toutes les réponses qu'ils reçoivent. Les paquets ainsi acceptés ne passeront même pas par les règles de filtrage.

EN PRATIQUE **ICMP et les états**

Quand un flux de données est autorisé par une règle **`keep state`**, IPFilter laissera passer tous ses paquets ainsi que les messages ICMP qui s'y rapportent. En particulier, les messages ICMP *source quench*, servant à la régulation du trafic, sont implicitement autorisés. Il n'est donc pas nécessaire de leur consacrer des règles spéciales.

Saturation de la table des états

Les filtres à états utilisent une table pour consigner les paquets sortants et pouvoir reconnaître les paquets entrants qui leur correspondent. Il est possible de la saturer : trop vaste, elle est longue à parcourir, et la latence augmente. Complètement remplie, on ne peut plus y ajouter de nouvelles entrées, et certains paquets sont rejetés à tort.

La taille par défaut de la table suffira pour un réseau de plusieurs centaines de machines. Au delà, les ajustements nécessiteront une recompilation du noyau. Aucune formule exacte ne permet de déterminer le nombre critique de machines qui oblige à faire cette modification, car tout dépend du trafic. En pratique, on commence par un noyau normal, et l'observation des troubles décrits au paragraphe précédent signalera qu'il faut agrandir la table. On peut aussi surveiller la table avec les commandes **`ipfstat -sl`** et **`ipfstat -t`**.

PLUS LOIN **Redimensionner la table des états**

C'est une opération un peu compliquée : pour NetBSD, il faut définir deux variables dans le fichier /sys/netinet/ip_state.h et recompiler le noyau (comme décrit au chapitre 13). Sous FreeBSD, la démarche est la même, mais le fichier à modifier s'appelle /sys/contrib/ipfilter/netinet/ip_state.h. Voici l'extrait concerné :

```
#ifndef IPSTATE_SIZE
# define        IPSTATE_SIZE    5737
#endif
#ifndef IPSTATE_MAX  /* Maximum number of states held */
# define        IPSTATE_MAX     4013
#endif
```

Le choix des bonnes valeurs est un exercice difficile à cause des contraintes imposées : ces deux nombres doivent être premiers, et `IPSTATE_MAX` vaudra environ 70 % de `IPSTATE_SIZE`. La commande **`primes`** (qui se trouve dans /usr/games) sera d'un grand secours : elle permet de générer des nombres premiers. Voici un petit script qui l'utilise pour produire des valeurs `IPSTATE_MAX` et `IPSTATE_SIZE` *ad-hoc* (on lui donne en argument la valeur minimale que l'on souhaite obtenir pour `IPSTATE_SIZE`) :

```
#!/bin/sh

test $# -ne 1 && echo "Usage: $0 target_value" && exit

begin=$1
end=`expr 2 \* $begin`
IPSTATE_SIZE=`/usr/games/primes $begin $end | head -1`

begin=`expr $IPSTATE_SIZE \* 70 / 100`
end=`expr 2 \* $begin`

IPSTATE_MAX=`/usr/games/primes $begin $end | head -1`

echo "IPSTATE_SIZE = $IPSTATE_SIZE"
echo "IPSTATE_MAX = $IPSTATE_MAX"
```

Bien faire le tri pour TCP

Une règle de filtrage à états pour TCP doit provoquer l'inscription d'une entrée dans la table dès le paquet d'initiation de la connexion. Elle permettra aux autres paquets de la même connexion de passer, qu'ils soient entrant ou sortants.

C'est pourquoi il n'est pas nécessaire d'autoriser les paquets ne correspondant pas à des initiations de connexion. On a vu au chapitre 9 que les paquets d'initiation de connexion TCP se distinguaient par leur drapeau SYN, non accompagné du drapeau ACK. On peut donc leur réserver une entrée dans la table des états ainsi :

```
pass in on ex0 proto tcp from any to any flags S/SA keep state
```

`flags S/SA` assure de ne créer des entrées dans la table que si nécessaire. Sur certaines anciennes versions d'IPFilter, cela était indispensable car en son absence une bogue (désormais corrigée) provoquait un remplissage rapide de la tables des états. Même sur des versions récentes du filtre, il vaut mieux prendre la bonne habitude de toujours accompagner **`keep state`** de **`flags S/SA`** pour les connexions TCP.

EN DÉTAIL **La syntaxe `flags S/SA`**

La syntaxe **`flags S/SA`** indique que de SYN (**`S`**) et ACK (**`A`**), seul SYN doit être actif.

VOCABULAIRE **Un bogue ou une bogue ?**

Bogue est la traduction française de l'anglais *bug*, qui littéralement se traduirait par « insecte ». L'origine du mot anglais vient d'une pionnière de l'informatique, qui, alors qu'elle cherchait la cause d'un dysfonctionnement, trouva une mite coincée dans un relais électrique à l'intérieur de la machine.
Les sonorités proches sont à l'origine de la traduction de *bug* par bogue, mais le genre du mot traduit reste incertain. Le masculin reste le plus employé, mais une parution au journal officiel recommande l'usage du féminin, comme pour l'enveloppe du marron. On devrait donc dire « une bogue ».

Du bon traitement des fragments

Autre problème : les paquets fragmentés. Par défaut, une règle **`keep state`** ne les traite pas. Les paquets fragmentés sortants vont donc créer des entrées supplémentaires dans la table, et les paquets fragmentés entrants seront bloqués. Si ce blocage se fait avec notification de type TCP RST (terminaison de la connexion), il sera difficile de télécharger des données.

Il faut donc tenir compte des fragments. Ceci se fait avec l'option **`keep frags`**, qu'on ajoutera systématiquement à toutes les règles **`keep state`** concernant TCP. C'est moins critique pour les protocoles UDP et ICMP, car ils sont rarement employés pour des paquets assez gros pour être fragmentés. Ce code n'en sera pas moins nécessaire pour transmettre de gros paquets UDP ou ICMP,

Reprenons notre exemple d'ACL en le modifiant pour utiliser des états. Nous n'autorisons désormais plus aucune communication initiée de l'extérieur, mais il est toujours possible d'initier des communications TCP, UDP, ou ICMP depuis le réseau interne.

PLUS LOIN **Qui est gros ?**

Un paquet sera fragmenté s'il est d'une taille supérieure au MTU du réseau sous-jacent (1500 octets sur Ethernet). Pour tester que le filtre laisse passer les paquets ICMP fragmentés, vous pouvez sur un réseau Ethernet tenter la commande suivante :

```
$ ping -s 1600 10.0.0.1
PING gizmo.local (10.0.0.1): 1600 data bytes
1608 bytes from 10.0.0.1: icmp_seq=0 ttl=255 time=10.284 ms
1608 bytes from 10.0.0.1: icmp_seq=1 ttl=255 time=10.250 ms
1608 bytes from 10.0.0.1: icmp_seq=2 ttl=255 time=10.386 ms
^C
----gizmo.local PING Statistics----
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 10.250/10.307/10.386/0.071 ms
$
```

```
block in return-icmp from any to any
block in return-rst proto tcp from any to any ❶

block in log quick from any to any with short ❷

block in return-rst quick proto tcp from any to any with ipopts
block in return-icmp quick from any to any with ipopts ❸

pass in quick on lo0 from any to any ❹

block in log quick from any to 127.0.0.0/8
block in log quick from 127.0.0.0/8 to any
block in log quick from 192.168.10.1/32 to any
block in log quick from 192.168.0.1/32 to any ❺

block in log quick on ex0 from !192.168.10.0/24 to any
block in log quick on ex0 from any to 192.168.10.0/24 ❻

block in log quick on fxp0 from 192.168.10.0/24 to any ❼

pass in on ex0 proto tcp from any to any flags S/SA keep state keep frags
pass in on ex0 proto udp from any to any keep state keep frags
pass in on ex0 proto icmp from any to any keep state keep frags ❽

pass in on fxp0 proto icmp icmp-type echo from any to any keep state keep frags ❾
```

❶ On commence par tout bloquer avec notification. Dans le cas général, on notifie par paquet ICMP, mais pour TCP on envoie un paquet RST. La règle pour TCP, plus spécifique, sera placée ensuite : c'est en effet la dernière règle qui l'emporte.

❷ On bloque les paquets trop courts pour être honnêtes, et on les consigne dans les journaux.

❸ Certaines options IP posent des problèmes de sécurité ; on les bloque donc par défaut. La règle pour TCP précède ici la règle générale : grâce au `quick`, elle va l'emporter.

❹ Tout le trafic sur `lo0` est autorisé.

❺ Grâce au `quick` pour le trafic sur `lo0`, les règles suivantes ne sont pas évaluées pour cette interface. On évacue ainsi un certain nombre de monstruosités, comme les paquets venant du réseau avec pour adresse source `127.0.0.1`.

❻ Sur `ex0` (interface interne), on bloque tout ce qui tente de sortir avec pour adresse source une adresse externe ou pour adresse de destination une adresse interne.

❼ Sur `fxp0`, on bloque de même ce qui prétend venir d'une adresse du réseau interne.

On a désormais évité l'*IP spoofing* contre le réseau interne ou le routeur lui-même.

❽ On autorise TCP, UDP et ICMP sortants et leurs réponses.

❾ On accepte les **`ping`** entrants, car il est malséant de les ignorer. Le **`keep state`** permet le passage des réponses.

Consignation des paquets

Le mot-clef `log` permet, on l'a vu, de consigner dans les journaux tout paquet suspect. En réalité, cela nécessite un peu de configuration.

IPFilter place ses journaux dans un espace de mémoire du noyau, accessible par le fichier spécial /dev/ipl. Le *daemon* **`ipmon`** a pour tâche d'en extraire les données

disponibles, qu'il peut renvoyer dans un fichier, ou transmettre à **syslogd** pour disposer de l'infrastructure standard de gestion des journaux.

ipmon peut être invoqué en ligne de commande, ou bien être invoqué par les scripts de démarrage (pour cela, il faut ajouter une ligne à /etc/rc.conf : **ipmon=YES** pour NetBSD, ou **ipmon_enable="YES"** pour FreeBSD).

Translation d'adresses

VOCABULAIRE **NAT**

En anglais, on parle de *Network Address Translator* (NAT). Les adeptes de GNU/Linux parlent souvent d'*IP masquerading*.

Cela consiste principalement à utiliser une plage d'adresses privées pour un réseau interne, relié à l'extérieur par un translateur d'adresses. Les machines du réseau privé peuvent communiquer avec l'extérieur, mais tous les paquets y sembleront émis par le translateur d'adresses.

Fonctionnement

Avant d'examiner les avantages et les inconvénients de la translation d'adresses, précisons un peu son fonctionnement.

Soit le translateur d'adresses Poulette, doté de deux interfaces. L'une, interne, a pour adresse `10.0.0.1/8` ; l'autre, externe, a pour adresse `192.0.2.13/24`. Dans le réseau interne, la machine Babar, d'adresse `10.0.0.3`, souhaite contacter le serveur Web du projet FreeBSD, d'adresse IP `216.136.204.117`. La figure 10.1 illustre ce fonctionnement :

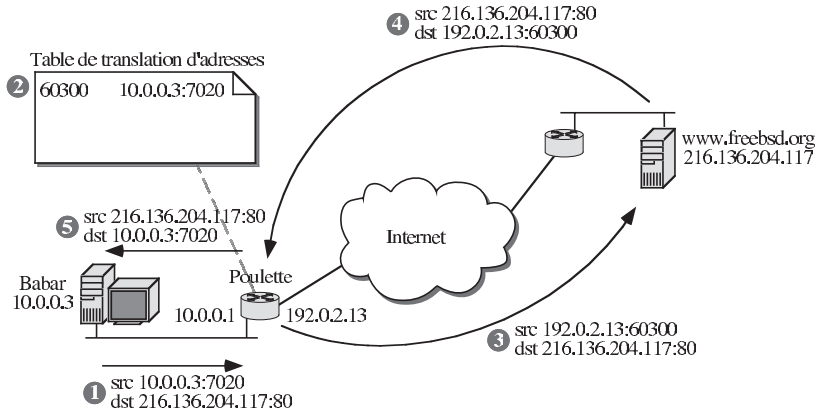


Figure 10–1 Fonctionnement d'un translateur d'adresses

NOTATION **IP et port**

`10.0.0.3:7020` signifie pour la source ou la destination d'un paquet, l'adresse IP `10.0.0.3` et le port TCP `7020`.

Babar émet un paquet TCP à destination de `www.freebsd.org`. Ce paquet a pour source `10.0.0.3:7020` et pour destination `216.136.204.117:80`. Poulette, passerelle par défaut de Babar, fait transiter le paquet.

Poulette transmet le paquet sur son interface externe en conservant son adresse de destination (`216.136.204.117:80`) mais en en modifiant l'adresse source :

192.0.2.13:60300. Poulette note dans une table que le numéro de port 60300 correspond à 10.0.0.3:7020.

Le paquet suit son chemin sur le réseau, parvient à www.freebsd.org, qui y répond. Le paquet de réponse, d'adresse source 216.136.204.117:80 et d'adresse destination 192.0.2.13:60300, fait sa route dans le réseau et parvient sur l'interface externe de Poulette.

Poulette consulte la table de translation d'adresses. Le port 60300 y correspond à 10.0.0.3:7020. Le paquet est donc transformé en conservant son adresse source, 216.136.204.117:80, mais en modifiant son adresse destination en 10.0.0.3:7020. Il est ensuite transmis sur le réseau interne : Babar reçoit enfin sa réponse, et le tour est joué.

La translation d'adresses : pour ou contre ?

La translation d'adresses donne la possibilité de pouvoir connecter autant de machines que l'on souhaite en ne disposant que d'une seule adresse IP publique. Dans une telle situation, c'est même la seule manière d'offrir une connectivité complète à plusieurs machines à la fois.

Autre avantage, côté sécurité : on utilise sur le réseau interne des adresses privées, non routées sur Internet. Un assaillant souhaitant s'introduire peinera : seul le translateur d'adresses peut atteindre le réseau privé. Pour s'introduire sur une de ses machines, il faudra donc d'abord s'inviter sur le translateur d'adresses.

Passons aux inconvénients. Sur le plan de la sécurité, si un utilisateur indélicat tente de s'introduire depuis le réseau interne vers un site extérieur, tentative détectée par l'administrateur du site en question, il donnera l'illusion que l'adresse d'où est parti le coup est celle du translateur d'adresses, et non pas celle du coupable. À moins de tenir des journaux exhaustifs du trafic réseau, on ne pourra pas tracer une activité suspecte jusqu'à sa source, situation assez inconfortable.

Autre problème : certains protocoles, tels que FTP ou *X Window System*, requièrent qu'une machine de l'extérieur se connecte sur une machine du réseau interne. La translation d'adresses rend ceci difficile, mais pas impossible, comme nous le verrons plus loin.

SÉCURITÉ **Translateur d'adresse et filtrage**

Le translateur d'adresses ne représente pas une sécurité absolue contre toute intrusion sur le réseau privé, même s'il est la seule machine pouvant y accéder, les autres routeurs en étant exclus. Il est impossible de contacter le réseau privé… sauf si on peut éviter ces routeurs.

En plaçant une machine sur le sous-réseau de l'interface externe du translateur d'adresses et en précisant ce dernier comme passerelle par défaut, on pourra initier des connexions vers le réseau interne : le translateur d'adresses routera docilement le trafic entre les deux réseaux.

La solution est simple : il suffit de filtrer un minimum sur le translateur d'adresses.

Certains services sur Internet authentifient ou limitent les accès par IP. Toutes les machines gérées par le translateur d'adresses apparaissant avec la même IP, il est possible que certains services ne puissent pas être utilisés par plusieurs machines du réseau interne à la fois. Le fournisseur de service concerné est fautif : il a mis en place un dispositif méconnaissant les problèmes de translation d'adresses. Malheureusement, c'est bien l'utilisateur qui en fera les frais, en étant banni du service.

Ce problème n'admet aucune solution simple hors de la négociation avec le fournisseur de services.

Mise en œuvre

Pour IPFilter, la translation d'adresses se configure dans le fichier /etc/ipnat.conf sur NetBSD, ou dans /etc/ipnat.rules sur FreeBSD. On fera prendre en compte toute modification avec la commande **`ipnat -FC -f /etc/ipnat.conf`**. Cette dernière détruira les règles existantes et rechargera les nouvelles depuis ce fichier.

La configuration de Poulette décrite précédemment se configure assez rapidement. En supposant qu'elle ait pour interface externe `ex0`, il suffit d'écrire les deux règles suivantes :

```
map ex0 10.0.0.0/8 -> 192.0.2.13/32 portmap tcp/udp 20000:65000
map ex0 10.0.0.0/8 -> 192.0.2.13/32
```

Ces deux règles sont nécessaires. L'une traite le trafic TCP et UDP, l'autre le reste, c'est-à-dire essentiellement le trafic ICMP. À la fin de la première ligne, on indique l'intervalle de ports à utiliser pour les ports source des paquets sortants du translateur d'adresses.

Erreur facile à commettre, ainsi qu'à éviter : la translation d'adresses doit être configurée sur l'interface externe. Si on la configure sur l'interface interne, les paquets issus du réseau privé sortiront sans transformation à l'extérieur, avec pour adresse source leurs adresses privées. Il est possible que ces paquets atteignent leur destination, mais les paquets de réponse ne pourront pas atteindre les machines, puisqu'ils auront pour destination des adresses privées qu'aucun routeur ne saura atteindre.

Enfin, on configure toujours un peu de filtrage sur un translateur d'adresses. Cela interdit tout accès au réseau interne depuis une machine du même sous-réseau que l'interface externe du translateur d'adresses. Cela évite aussi toute fuite de paquets portant des adresses privées en cas de désactivation ou de mauvaise configuration de la translation d'adresses.

Lors de la conception de tels filtres, il faut garder à l'esprit que sur une interface donnée, les traitements de translation d'adresses sont toujours faits avant le filtrage en entrée et après lui en sortie. Un paquet parvenant sur l'interface externe à destination du réseau interne présentera au filtre comme adresse de destination l'adresse de la machine du réseau interne, et non pas l'adresse externe du translateur d'adresses.

RAPPEL **Initialisation de IPFilter**

Pour activer la translation d'adresses sans passer par les scripts de démarrage, il faut d'abord initialiser IPFilter avec **`ipf -E`**
Pour obtenir l'initialisation automatique de la translation d'adresses au démarrage, il faut ajouter une ligne au /etc/rc.conf : **`ipnat=YES`** pour NetBSD, et **`ipnat_enable="YES"`** pour FreeBSD.

ATTENTION **`portmap`**

Le mot-clef **`portmap`** dans IPFilter n'a rien à voir avec le *daemon* **`portmap`** utilisé pour les services RPC.

ATTENTION **Saturation de la table de translation d'adresses**

Tout comme la table des états, la table du translateur d'adresses est susceptible d'être saturée lorsqu'elle opère sur des centaines de machines. Sur NetBSD, on corrige cela dans le fichier /sys/netinet/ip_nat.h, où l'on ajustera la macro NAT_SIZE avant de recompiler le noyau. Sur FreeBSD, le fichier ip_nat.h se trouve dans /sys/contrib/ipfilter/netinet/.

Protocoles à problèmes

On l'a déjà évoqué, tout protocole impliquant une connexion issue de l'extérieur pose problème ; c'est notamment le cas de FTP. *X Window System* gênera aussi, dans une moindre mesure, ainsi que tous les logiciels de visio-conférence ou de communication pair à pair (*peer-to-peer*).

Problèmes de clients FTP

FTP est un protocole exagérément complexe, qui laisse perplexe sur l'état d'esprit de ses concepteurs. Lors d'une session FTP, le client fait une première connexion, dite de contrôle, vers le serveur. Cette connexion part d'un port quelconque du client et aboutit sur le port 21 du serveur.

Elle sert tant qu'aucun fichier n'est échangé : elle permet par exemple de se déplacer dans l'arborescence du serveur, de créer ou supprimer des répertoires, de supprimer des fichiers.

En revanche, toute requête d'affichage du contenu d'un répertoire (forme de transfert d'un contenu de fichier), tout envoi ou toute réception d'un fichier, imposent l'utilisation d'une deuxième connexion, dite de données. Deux cas sont alors possibles.

Dans la méthode originale, dite FTP actif, le serveur initie une connexion TCP de son port 20 vers un port quelconque du client, négocié dynamiquement par le biais de la connexion de contrôle. Cette méthode ne fonctionnera évidemment pas si le client est inaccessible à cause d'un filtre ou d'un translateur d'adresses.

C'est la raison d'être du FTP passif, où le client initie la connexion de données, depuis un port quelconque vers un port qui lui est indiqué par le serveur. Cette méthode n'étant en rien gênée par les filtres et translateurs d'adresses placés côté client, on pourrait être tenté de la préférer systématiquement. Hélas, plus récente que le FTP actif, elle n'est pas implémentée par les clients les plus anciens. Elle fait par exemple défaut au client FTP en mode DOS de Windows 95/98/Me et NT. De plus, certains serveurs FTP ne fonctionnent qu'en mode actif.

Il est donc souvent obligatoire de passer par le FTP actif. La seule manière de procéder est d'utiliser un mandataire (en anglais *proxy*), programme jouant le rôle d'intermédiaire entre client et serveur. Placé sur le translateur d'adresses, il sera accessible depuis le serveur, et le FTP actif sera possible... si le client daigne utiliser un mandataire. Malheureusement, les clients incapables de comprendre le FTP passif sont en général également incapables d'utiliser un mandataire. La solution, c'est le mandataire transparent (*proxy* transparent).

Un mandataire transparent intercepte le trafic et se comporte en mandataire à l'insu du client. IPFilter en contient un, spécialement conçu pour les questions de FTP passif. Pour l'utiliser dans la configuration décrite précédemment, il suffit d'ajouter une ligne à `/etc/ipnat.conf` (attention, l'ordre des lignes est important) :

```
map ex0 10.0.0.0/8 -> 192.0.2.13/32 portmap tcp/udp 20000:65000
map ex0 10.0.0.0/8 -> 192.0.2.13/32 proxy port ftp ftp/tcp
map ex0 10.0.0.0/8 -> 192.0.2.13/32
```

Problème de serveur FTP

Un serveur FTP placé derrière un filtre pose d'autres problèmes : le fonctionnement du FTP passif impose d'autoriser les connexions de l'extérieur, tant sur le port 21 (connexion de contrôle) que sur tous les ports utilisés pour les connexions de données. Négociés par défaut, ces ports sont peu contraints (intervalle 1024 à 65535).

Si cela semble trop permissif, certains serveurs FTP peuvent restreindre cette plage de ports (c'est en effet le serveur qui impose le port au client pour la connexion de données). Dans l'exemple du serveur FTP de la distribution de base des systèmes BSD, c'est l'option **`portrange`** qui apporte cette fonctionnalité. La page **`man`** de `ftpd.conf` apportera tous les détails.

Problèmes avec *X Window System*

X Window System permet des sessions en mode graphique à distance sur une machine Unix, à l'image des sessions texte à distance des commandes **`telnet`** ou **`ssh`**. Différence de taille : dans le cadre du protocole X, l'écran d'affichage (la machine locale) est le serveur, auquel le client (la machine distante) doit pouvoir se connecter.

Toute session *X Window System* commence par une connexion sur la machine distante par **`telnet`**, **`ssh`**, ou via un protocole de gestion de session X tel que XDMCP. On y exécute les clients X, automatiquement ou à la main, et sous réserve d'une variable `DISPLAY` bien positionnée, ils se connectent au serveur X de la machine locale pour afficher leurs interfaces graphiques.

Le même problème que pour le FTP actif se pose. Pas besoin toutefois de mandataire si l'on se connecte avec **`ssh`**. C'est en effet une bonne méthode pour se connecter à un site distant, car le trafic traverse le réseau chiffré. On évite ainsi toute interception de la session, et notamment des mots de passe ou autres informations confidentielles qui y transitent. Mais **`ssh`** est de plus capable de construire des tunnels dans la connexion existante pour y faire passer d'autres connexions, reliant client et serveur SSH.

En résumé, **`ssh`** s'occupe de tout. Invoqué depuis un environnement *X Window System*, il crée un tunnel du port `n` de la machine distante au port 6000 de la machine locale et fixe la variable `DISPLAY` dans la session distante pour que ses clients X puissent contacter le serveur à travers le port `n` distant.

VOCABULAIRE **XDMCP**

XDMCP (*X Display Manager Control Protocol*, ou protocole de contrôle du gestionnaire d'affichage X) est un protocole utilisé par le *daemon* **`xdm`** lorsque le client et le serveur sont situés sur des machines différentes.

CULTURE **Serveur X**

Une session *X Window System* nécessite l'emploi sur la machine locale d'un logiciel appelé serveur X. Il accepte les connexions du client X et se pliera à ses demandes d'affichage. On l'appelle parfois à tort « émulateur *X Window* ».
Le serveur X est habituellement en écoute sur le port TCP 6000. Les clients qui doivent le contacter utilisent la variable d'environnement **`DISPLAY`**, de syntaxe **`host:0.0`**. **`host`** est l'adresse du serveur X à contacter (la machine locale par défaut). Le premier **`0`** indique d'utiliser le port 6000 (**`host:2.0,`** indiquerait d'utiliser le port 6002 ; on peut ainsi employer plusieurs serveurs X sur la même machine). Le deuxième zéro indique le numéro d'écran à utiliser si le serveur X en gère plusieurs. Il est facultatif ; **`host:0`** suffit.

Les requêtes X émises en provenance de ce dernier traversent le tunnel et aboutissent sur le serveur X de la machine locale. On évite ainsi les problèmes de filtrage, et on bénéficie d'un trafic chiffré.

Parfois, pour une raison quelconque, cela ne fonctionne pas. Il faut alors faire le travail à la main. Exemple :

```
local$ ssh -R6010:localhost:6000 remote
remote$ export DISPLAY=localhost:10
remote$ xterm &
```

La page **man** de **ssh** détaillera l'option **-R**. En bref, on installe un tunnel du port 6010 de la machine distante au port 6000 de la machine locale, où le serveur X est en écoute.

Problèmes de visio-conférence et de *peer-to-peer*

Sans mandataire, une partie des fonctionnalités du logiciel seront indisponibles. Dans le cas des communications pair à pair (*peer-to-peer*), le filtre empêchera tout téléchargement distant de fichiers locaux, mais pas l'inverse.

IPFilter dispose d'un mandataire transparent pour **rsh**, **rexec**, et **rlogin**, qui se configure comme le *proxy* FTP en remplaçant **ftp** par **rcmd**. Il dispose aussi d'un mandataire pour les flux *Real Audio* : **raudio**, et d'un mandataire H323 : "**h323**". Les autres protocoles nécessiteront des mandataires externes, que nous évoquerons dans la section suivante.

CULTURE **rsh, rexec et rlogin**

rsh, **rexec** et **rlogin** facilitaient les interactions entre machines Unix. **rsh** et **rexec** permettaient d'exécuter des commandes à distance, et **rlogin** d'invoquer une session sur une machine distante. Des mécanismes aujourd'hui réputés pour leur insécurité (fichiers .rhosts) évitaient de fournir un mot de passe pour utiliser ces commandes, que **ssh** remplace toutes trois avantageusement – c'est pourquoi nous en avons parlé à l'imparfait.

CULTURE **H323**

H323 est un protocole utilisé pour les visio-conférences point à point. Il provient directement de la visio-conférence sur RNIS.

Redirection de ports, mandataires applicatifs

Parfois, il faut rendre accessible à l'extérieur un service placé derrière un translateur d'adresses. On peut aussi souhaiter rediriger un flux sur une autre machine que celle initialement prévue. La solution est la même : la redirection de ports.

Avec IPFilter, la configuration se place dans `/etc/ipnat.conf`. Dans l'exemple précédent, on rendra accessible par le port 80 du translateur d'adresses le port 8080 de Babar (il y héberge un serveur Web) en écrivant une règle **rdr** :

```
map ex0 10.0.0.0/8 -> 192.0.2.13/32 portmap tcp/udp 20000:65000
map ex0 10.0.0.0/8 -> 192.0.2.13/32 proxy port ftp ftp/tcp
map ex0 10.0.0.0/8 -> 192.0.2.13/32

rdr ex0 192.0.2.13/32 port 80 -> 10.0.0.3 port 8080 tcp
```

Les règles **rdr** sont très puissantes, et permettent notamment de répartir la charge. En présence d'un deuxième serveur, d'IP `10.0.0.5` et hébergeant une réplique du serveur Web de Babar, on pourrait répartir les requêtes comme suit :

```
map ex0 10.0.0.0/8 -> 192.0.2.13/32 portmap tcp/udp 20000:65000
map ex0 10.0.0.0/8 -> 192.0.2.13/32 proxy port ftp ftp/tcp
map ex0 10.0.0.0/8 -> 192.0.2.13/32

rdr ex0 192.0.2.13/32 port 80 -> 10.0.0.3,10.0.0.5 port 8080 tcp round-robin
```

Abordons enfin le cas de la redirection vers un mandataire applicatif. On peut détourner tout le trafic issu du réseau interne et destiné au port 80 (Web) sur un mandataire applicatif tel que Squid, situé sur le translateur d'adresses, et écoutant sur le port 3128.

CULTURE **Squid**

Squid, disponible dans le système de paquetages des systèmes BSD, est un mandataire pour plusieurs protocoles, dont HTTP et FTP.

```
map ex0 10.0.0.0/8 -> 192.0.2.13/32 portmap tcp/udp 20000:65000
map ex0 10.0.0.0/8 -> 192.0.2.13/32 proxy port ftp ftp/tcp
map ex0 10.0.0.0/8 -> 192.0.2.13/32

# Rappel: fxp0 est l'interface interne de Poulette...
rdr fxp0 0.0.0.0/0 port 80 -> 127.0.0.1 port 3128 tcp
```

Les possibilités sont très étendues. On peut utiliser Squid en cache (on dit aussi antémémoire : c'est une copie locale conservée pour un accès plus rapide), obligeant ainsi les utilisateurs à utiliser un cache collectif à leur insu, quels que soient les réglages de leur navigateur. Squid est également capable de filtrage : on pourra donc interdire certains contenus, comme les films de plus de 10 Mo, par souci d'économiser la bande passante.

Un mandataire applicatif pourra encore filtrer les virus, laissant ainsi les chevaux de Troie et autres indélicatesses du Web à la porte.

De IPFilter à PacketFilter

PacketFilter remplace IPFilter dans OpenBSD 3.0. Officiellement, c'est à cause d'un problème de licence d'IPFilter, trop floue au goût des développeurs du projet OpenBSD. En réalité, il semble que des problèmes relationnels entre certaines personnes soient à l'origine de ce nouveau schisme.

Nouveauté ou stabilité ?

IPFilter et PacketFilter réalisent environ les même tâches. Proches dans leur conception, ils ne sont pas intégrés aux systèmes BSD de la même façon. La version d'IPFilter qu'on trouve dans les distributions de NetBSD et FreeBSD n'est pas la dernière ; elle est issue d'une branche stable, sur laquelle seuls les bogues ont été corrigées.

PacketFilter pour sa part bénéficie constamment d'un apport de nouvelles fonctionnalités : il est dépourvu de branche « stable ». Cette différence dans le mode de

DÉBAT **Lequel est le meilleur ?**

Encore une question qu'il ne faut pas poser en public sous peine de déclencher une guerre de tranchées entre les partisans des deux filtres. Retenez l'essentiel : selon toute vraisemblance, les deux vous conviendront.

développement a un impact pour l'utilisateur : PacketFilter est plus agressif dans ses évolutions, propose plus de fonctionnalités, plus rapidement. IPFilter est plus stabilisé : toute bogue sera probablement corrigée dans une version plus récente de la branche stable... mais celle-ci n'inclura pas de nouvelles fonctionnalités.

Appliquer le même raisonnement dans PacketFilter amènera à installer une version récente, corrigeant la bogue, mais proposant également des nouvelles fonctionnalités, et donc probablement des nouvelles bogues.

Autre différence importante : l'interface utilisateur de PacketFilter n'est pas figée, ce qui peut obliger l'administrateur à retoucher sa configuration lors des mises à jour. IPFilter assure pour sa part une compatibilité ascendante dans ce domaine.

PLUS LOIN **IPFilter sur OpenBSD et PacketFilter sur NetBSD ou FreeBSD**

Les choix du filtre et du système d'exploitation sont indépendants. Les distributions officielles des systèmes BSD proposent IPFilter/FreeBSD, IPFilter/NetBSD, ou PacketFilter/OpenBSD, mais on peut très bien installer PacketFilter sur NetBSD ou FreeBSD tout comme on peut faire fonctionner IPFilter sur OpenBSD : il suffit d'installer soi-même le filtre. L'opération peut éventuellement être complexe, mais elle reste du domaine du possible.

ATTENTION **Documentation à jour**

PacketFilter évolue vite. Le format du fichier de configuration évolue, il faut donc utiliser la documentation à jour. Ce livre vous donne un aperçu des possibilités de configuration de PacketFilter, mais pour passer à la mise en œuvre proprement dite, la lecture de la documentation en ligne – la seule à jour – sera probablement indispensable.

Configuration de PacketFilter

Le projet OpenBSD propose une documentation très complète sur PacketFilter à l'adresse http://www.openbsd.org/faq/pf. Les ACL sont les mêmes qu'avec IPFilter : il s'agit toujours de bloquer ou de laisser passer des paquets, mais la syntaxe et les commandes changent un peu.

Activation et désactivation du filtre

Le filtre s'active avec **`pfctl -e`**. On l'active au démarrage en plaçant **`pf=YES`** dans le fichier /etc/rc.conf.

CULTURE **Les licences**

Les logiciels libres ne sont pas tous distribués sous la même licence. La plus répandue, la licence publique générale de GNU (*GNU General Public License*, ou GPL), créée par la *Free Software Foundation* (FSF), sert notamment au noyau Linux et aux logiciels du projet GNU. Elle fournit les quatre libertés du logiciel libre et stipule essentiellement que toute redistribution à un tiers du logiciel du logiciel ou d'une version modifiée (œuvre dérivée) doit lui fournir les mêmes droits, à sa demande. Elle impose donc que toute œuvre dérivée soit à son tour distribuée sous GPL : c'est le principe du « gauche d'auteur ». Ses détracteurs l'accusent pour cela d'être un virus, ce que la FSF conteste : c'est un terme péjoratif et le but premier et principal est de protéger et de libérer les utilisateurs.

La licence publique générale réduite de GNU (*GNU Lesser General Public License*, ou LGPL), similaire à la GPL, n'impose pas forcément la fourniture du code source de logiciels liés dynamiquement à un produit placé sous LGPL. Elle est surtout utilisée pour les bibliothèques dont on ne souhaite pas entraver l'utilisation dans des logiciels propriétaires. C'était une manœuvre stratégique de la FSF, qui recommande désormais de lui préférer la GPL, la masse critique de logiciel libre étant atteinte.

La licence BSD adopte une autre approche : elle exige simplement que la provenance du code source et l'identité de son auteur soient indiquées lors de la redistribution ou de la distribution d'une version modifiée. L'utilisation du nom de l'auteur à des fins de promotion sans son accord est également interdite. En particulier, il est possible de modifier un produit sous licence BSD et de distribuer l'œuvre ainsi dérivée sans être tenu de fournir de code source – « propriétarisant » par là le logiciel. Microsoft et Apple ont utilisé cette possibilité dans certains de leurs produits, parmi les plus prestigieux.

Cette « propriétarisation » ne pose pas de problèmes aux développeurs utilisant la licence BSD : l'œuvre originale reste utilisable en licence BSD, et ils ont la satisfaction d'avoir injecté du code de bonne qualité dans un logiciel propriétaire. De plus, il arrive souvent que les entreprises publient en licence BSD les améliorations qu'elles apportent à une œuvre distribuée sous licence BSD, même si elles n'y sont pas légalement contraintes : cela leur évite d'avoir à maintenir des versions parallèles des sources, et leur permet de profiter de l'innovation en provenance de la communauté *Open Source*. Apple a très largement embrassé cette pratique avec le développement des couches Unix de MacOS X.

Il existe de nombreuses autres licences libres ou *Open Source*, dont on trouvera respectivement des catalogues sur le site Web du projet GNU (http://www.gnu.org/philosophy/license-list.fr.html) ou de l'*Open Source Initiative* (http://opensource.org/licenses).

La configuration du filtre se trouve dans le fichier /etc/pf.conf, son comportement se contrôle et surveille avec la commande **pfctl**. Ainsi, **pfctl -f /etc/pf.conf** chargera les règles.

ASTUCE **Vérifier /etc/pf.conf**

La commande **pfctl -nf /etc/pf.conf** permet de contrôler la syntaxe du fichier /etc/pf.conf sans le charger pour autant. Elle permet d'éviter la situation où une erreur dans le fichier cause la désactivation d'une partie du dispositif de filtrage.

Le fichier de configuration de PacketFilter

/etc/pf.conf contient différents types d'instructions :

- définition des options ;
- définition des variables et des listes ;
- règles de filtrage (équivalent du /etc/ipf.conf d'IPFilter) ;
- règles de translation d'adresses et de redirection (équivalent du /etc/ipnat.conf d'IPFilter).

Options

Les options indiquent le comportement général de PacketFilter : on lui précise la quantité de mémoire à allouer à la table d'états et à la table des fragments, le type d'optimisation, les temps d'expiration, et le comportement par défaut des règles **block**.

Exemple de déclaration d'options, extraites de la documentation de PacketFilter :

```
set timeout interval 10
set timeout frag 30
set limit { frags 5000, states 2500 }
set optimization high-latency
set block-policy return
set loginterface dc0
```

EN PRATIQUE **Tables d'états et de fragments**

C'est la même table d'états que dans le cas d'IPFilter. Ce dernier impose de recompiler le noyau pour la redimensionner, alors que PacketFilter sait faire cette opération à chaud.
La table des fragments, spécifique à PacketFilter, est utilisée par les règles **scrub** (dont nous parlerons plus loin).

Listes et macros

IPFilter impose souvent de répéter plusieurs fois la même règle de filtrage pour plusieurs machines ou plusieurs ports différents. Pour autoriser par exemple l'entrée des protocoles SMTP, POP et IMAP sur un serveur de messagerie, on écrira :

```
pass in on ex0 from any to 192.0.2.154 port = 25 keep state keep frags
pass in on ex0 from any to 192.0.2.154 port = 110 keep state keep frags
pass in on ex0 from any to 192.0.2.154 port = 143 keep state keep frags
```

PacketFilter permet de factoriser cela :

```
pass in on ex0 from any to 192.0.2.154 port { 25, 110, 143 } keep state keep frags
```

EN PRATIQUE **Comportement de `block`**

Avec IPFilter, on peut indiquer dans chaque règle **block** si on souhaite le renvoi d'un paquet ICMP (**block in return-icmp** ...), le renvoi d'un paquet TCP avec drapeau RST (**block in return-rst** ...), ou un blocage sans notification (**block in** ...).
En indiquant **set block-policy return** dans /etc/pf.conf, on indique à PacketFilter que chaque règle **block** devra produire le renvoi d'un paquet ICMP ou TCP RST. **set block-policy drop** demande au contraire de bloquer silencieusement les paquets.

ATTENTION **Spécification des ports TCP et UDP**

IPFilter utilise le signe **=** (égal) pour spécifier un port TCP ou UDP, alors que PacketFilter se contente d'un blanc. En revanche, ces deux filtres précisent les intervalles de ports de la même manière, avec les symboles **>**, **<**, **><** et **<>**.

Une seule règle regroupe les trois ouvertures de ports. La terminologie de PacketFilter appelle « listes » de tels groupements ; on peut les utiliser pour préciser des ports, des adresses IP, ou des protocoles de niveau 4 (exemple : **`proto { tcp, udp, icmp }`**).

PacketFilter offre aussi la possibilité d'écrire des macros en début de fichier, macros ensuite développées dans les règles de filtrage. On peut donc écrire :

```
ext = "ex0"
mail_server = "192.0.2.154"

pass in on $ext from any to $mail_server port { 25 110 143 } keep state keep frags
```

C'est extrêmement intéressant : si le fichier de règles est bien écrit, toute modification de la configuration des cartes du pare-feu ne devra être répercutée qu'au seul niveau des définitions de macros.

De même, si l'adresse du serveur de messagerie change, il suffira d'en préciser la nouvelle adresse dans une seule ligne du fichier. C'est une bonne habitude en programmation et en informatique que de n'écrire qu'une fois chaque information.

Les listes et macros améliorent considérablement la lisibilité des règles de filtrage. Mais PacketFilter propose encore plus.

ASTUCE **Simplifications**

Les virgules sont facultatives. Ainsi, les deux règles suivantes sont équivalentes :

```
pass in on ex0 from any to 192.0.2.154 port { 25, 110, 143 } keep state keep frags
pass in on ex0 from any to 192.0.2.154 port { 25 110 143 } keep state keep frags
```

Normalisation des paquets : fonction `scrub`

Nous avons abondamment abordé la fragmentation des paquets IP : elle survient quand un paquet est trop gros pour pouvoir traverser un réseau. Après fragmentation, les paquets finissent leur voyage tels quels, en subissant parfois d'autres fragmentations. C'est en principe la machine destination qui les rassemble.

Dans le passé, on a trouvé des bogues dans le code chargé de rassembler les paquets. En 1998, l'attaque Teardrop utilisait des fragments IP mal formés pour planter les noyaux des systèmes vulnérables. En principe, tous sont désormais corrigés, mais on trouve toujours des postes antédiluviens que personne n'a jamais mis à jour. Et qui sait ? un autre bogue du même type fera peut-être son apparition.

PacketFilter propose donc une fonction de normalisation des paquets sur le pare-feu : **`scrub`**. Les fragments sont contrôlés et rassemblés. La machine destinataire ne risque donc plus une attaque par fragments mal formés. **`scrub`** protège contre certains dangers, à condition bien sûr que le pare-feu lui-même n'y succombe pas... Ce n'est pas impossible : en 1998, OpenBSD était vulnérable à l'attaque Teardrop.

PLUS LOIN **`scrub` et la table des fragments**

Avec **`scrub`**, PacketFilter doit maintenir une table de fragments en cours d'assemblage. Si elle sature, on peut ajuster sa taille avec l'option **`set limit`**.

scrub s'active par une ligne telle que :

```
scrub in all fragment reassemble
```

PIÈGE À C... **Les paquets se perdent ?**
scrub présente quelques problèmes d'interopérabilité, notamment avec certaines implémentations de NFS ou avec des jeux en réseau. Toute perte de paquets dans une telle situation lui est potentiellement imputable. La documentation de PacketFilter détaille ce point.

Translation d'adresses et redirection

La translation d'adresses se configure elle aussi dans /etc/pf.conf. La syntaxe diffère légèrement de ce qui se pratique avec IPFilter, puisqu'une seule ligne résume les deux qui lui sont nécessaires ;

```
nat on rtk1 from 10.0.0.0/8 to any -> 192.0.2.15
```

La redirection de ports, semblable en principe à celle d'IPFilter, en diffère aussi par la syntaxe. Exemple :

```
rdr fxp0 proto tcp from any to any port 80 -> 127.0.0.1 port 3128
```

Gestion de FTP

On l'a mentionné, FTP a besoin d'un mandataire pour laisser passer le FTP en mode actif. IPFilter fournit un *proxy* transparent placé au sein même du noyau. PacketFilter fait lui appel à un mandataire externe en espace utilisateur.

Pour faire fonctionner FTP en mode actif vers des clients situés derrière un pare-feu sous PacketFilter, il faudra donc rediriger tout le trafic FTP sur ce mandataire avec une règle comme celle ci :

```
rdr on fxp0 proto tcp from any to any port 21 -> 127.0.0.1 port 2121
```

Le mandataire FTP se trouve sous OpenBSD dans /usr/libexec/ftp-proxy. Il est conçu pour être invoqué par **inetd**, en présence d'une ligne comme suit dans le fichier /etc/inetd.conf :

```
127.0.0.1:2121 stream tcp nowait root /usr/libexec/ftp-proxy ftp-proxy
```

RAPPEL **Fonctionnement de `inetd`.**
Toute modification du fichier /etc/inetd.conf ne sera prise en compte qu'après envoi du signal SIGHUP (**`kill -1`**) à **inetd**.

PIÈGE À C... **FTP passif et serveur FTP**
Tout serveur FTP placé derrière le pare-feu nécessitera pour les connexions de données d'autres ouvertures de ports que celle du port 21. Les ports à ouvrir dépendent de la configuration du serveur FTP – problème traité précédemment.

PLUS LOIN **NAT et adresses dynamiques**
PacketFilter apporte de la souplesse. On peut par exemple indiquer le nom de l'interface extérieure au lieu de son adresse IP : si cette dernière change, le fichier de configuration sera toujours valable :

```
nat on rtk1 from 10.0.0.0/8 to any -> rtk0
```

L'adresse IP à utiliser pour l'interface externe est utilisée à la configuration de PacketFilter. Si celle-ci doit changer souvent (cas des configurations utilisant PPPoE, PPP, ou DHCP), on peut indiquer à PacketFilter de la maintenir à jour en mettant le nom de l'interface entre parenthèses :

```
nat on rtk1 from 10.0.0.0/8 to any -> (rtk0)
```

Contrôle de la qualité de service

La qualité de service concerne les limitations ou réservations de bande passante par catégorie d'utilisateurs ou de services, afin d'éviter qu'une minorité ne consomme les ressources réseau au détriment du plus grand nombre.

Avec IPFW, FreeBSD dispose depuis longtemps d'un dispositif de limitation de bande passante, grâce à sa fonctionnalité **`dummynet`**. NetBSD et OpenBSD étaient jusqu'à récemment dépourvus d'un tel outil. Désormais, les trois BSD proposent un dispositif très complet de contrôle de la qualité de service : ALTQ.

ALTQ est un sous-projet du projet KAME, visant originellement à intégrer une pile IPv6 dans les trois systèmes BSD. Il implémente une foule de stratégies de gestion de la bande passante : BLUE, CBQ, FIFOQ, HFSC, PRIQ, RED, RIO, WFQ, et CDNR. Leur description exhaustive sort du cadre de ce chapitre, mais elles sont détaillées dans la page **`man`** de altq.conf.

SUR LES AUTRES UNIX **Contrôle de la qualité de service**

D'autres systèmes fournissent des fonctionnalités équivalentes. Par exemple, le projet *Linux Traffic Control* (*Linux TC*) implémente quelques stratégies de gestion de bande passante dans GNU/Linux.

PLUS LOIN **Stratégies de gestion de bande passante**

Sans entrer les détails, voici les rôles des différentes stratégies gérées par ALTQ.

FIFOQ	*First-In First-Out Queue* : les paquets sortent dans l'ordre d'entrée. C'est le cas le plus simple, qui correspond à l'absence de gestion de qualité de service.
PRIQ	*PRIority Queueing* : stratégie qui permet de définir des classes de trafic plus ou moins prioritaires. Les paquets de classes moins prioritaires ne pourront pas passer tant qu'il restera des paquets d'une classe plus prioritaire en attente. Cette méthode permet par exemple d'assurer une priorité maximum aux sessions interactives SSH, au détriment des téléchargements en FTP.
WFQ	*Weighted Fair Queueing* : chaque flux dispose d'une file d'attente individuelle, que le routeur tente de traiter équitablement. On peut ajouter une pondération sur certains flux pour qu'ils bénéficient d'un traitement de faveur.
SFQ	*Stochastic Fairness Queueing* : comme WFQ, avec des files d'attente partagées entre flux. La bande passante est partagée moins équitablement, mais le routeur consomme moins de ressources (mémoire et processeur) en présence de nombreux flux différents.
CBQ	*Class Based Queueing* : stratégie qui permet de définir des classes hiérarchisées de trafic disposant chacune d'un quota de bande passante. Idéal pour répartir cette dernière entre différentes entités d'un établissement.
HFSC	*Hierarchical Fair Service Curve* : CBQ amélioré : on peut de plus distinguer les besoins en fort débit des besoins en faible latence.
RED	*Random Early Detection* : provoque le rejet de paquets des files d'attentes trop longues. Au lieu de rejeter tous les paquets une fois que la file est pleine, on en rejette quelques-uns au hasard pour essayer de la maintenir courte, dans le but de minimiser les effets de la congestion sur les connexions TCP.
BLUE	Jeu de mots : le bleu (*blue*) s'oppose au rouge (*red*) de la stratégie RED. Cette stratégie vise les mêmes objectifs, mais en utilisant les pertes de paquets et l'activité des liens pour déterminer s'il faut commencer à rejeter des paquets, dans le but d'obtenir de meilleures performances qu'avec RED.
RIO	*RED with In/Out* : un RED discriminant les paquets sortants dans sa gestion de la congestion.
JoBS	*JOint Buffer management and Scheduling* : algorithme visant à donner des garanties absolues et relatives sur la qualité de service offerte à différentes classes de trafic.
CDNR	*CoNDitioneR* : pas à proprement parler une stratégie de gestion de la qualité de service. Sert à rejeter, marquer, ou compter des paquets.

Mise en œuvre d'ALTQ

ALTQ étant récent, il n'est pas forcément activé dans le noyau installé par défaut. Il faudra parfois recompiler un noyau avec ALTQ et différentes stratégies de gestion de la bande passante.

Sous OpenBSD, ALTQ se configure depuis le fichier de configuration de PacketFilter, ce qui est très pratique. La même fusion est prévue pour IPFilter, mais dans l'intervalle, il faut écrire un fichier de configuration séparé : /etc/altq.conf. Nous allons détailler cette voie.

/etc/altq.conf est lu par le *daemon* **altqd**. Toute modification du fichier ne sera prise en compte qu'après envoi du signal SIGHUP (**kill -1**) à ce dernier.

Dans le cas du réseau déjà décrit. Poulette et liée à l'extérieur par un lien de 2 Mb/s de débit descendant. Babar dispose d'un serveur Web sur le port 8080, dont nous voulons limiter le trafic Web entrant à 40 % de la bande passante totale, réservant ainsi les 60 % restants aux autres machines. Elles pourront profiter d'un éventuel complément de bande passante si Babar n'exploite pas toute sa part, mais ce dernier ne pourra pas monter au-delà de 40 %.

On réalise cela avec la stratégie CBQ (*Class Based Queing* : files d'attente par classes). Deux classes sont utilisées : l'une pour Babar, l'autre pour les autres machines. Voici les règles à inscrire dans /etc/altq.conf :

```
interface fxp0 bandwidth 2M cbq
class cbq fxp0 class_root NULL pbandwidth 100

class cbq fxp0 class_babar class_root pbandwidth 40
class cbq fxp0 class_autre class_root borrow pbandwidth 60 default

filter fxp0 class_babar 0 0 10.0.0.3 netmask 255.255.255.255 8080 6
filter fxp0 class_autre 0 0 10.0.0.0 netmask 255.0.0.0 0 0
```

Quelques commentaires : d'abord, l'interface indiquée est l'interface interne de Poulette, car on souhaite contrôler le trafic pénétrant le réseau interne. ALTQ n'agit en effet que sur les paquets sortants du routeur. Pour contrôler le trafic sortant du réseau interne, il aurait fallu manipuler l'interface externe de Poulette.

Les clauses **interface** et **class** sont assez simples à comprendre. Chaque classe porte un nom et a une autre classe pour parent, sauf la classe racine (**root**), dont le parent est NULL. On crée ici deux classes, disposant respectivement de 40 % et

EN PRATIQUE **ALTQ et PacketFilter**

Avec PacketFilter, cette configuration CBQ peut être précisée via /etc/pf.conf comme suit :

```
altq on fxp0 cbq bandwidth 2Mb queue { babar, autre }

queue babar bandwidth 40%
queue autre bandwidth 60% cbq(borrow default)

pass out on fxp0 from any to 10.0.0.3 port 8080 queue babar
pass out on fxp0 from any to 10.0.0.0/24 queue default
```

La syntaxe est assez intuitive : la ligne **altq** indique la stratégie, la bande passante, et les classes filles de la classe racine (ici, **babar** et **autre**).
Les lignes **queue** indiquent les propriétés des classes : bande passante et options pour la stratégie de gestion de la qualité de service. On donne ici les options **borrow** et **default** pour CBQ.
Enfin, il ne reste plus qu'à préciser à quelles classes appartiennent les paquets, dans les règles **pass**.

60 % du trafic, quantités indiquées par le mot-clef **pbandwidth**. **borrow** précise qu'une classe peut emprunter du trafic non utilisé à sa classe parent. **default** dénote une classe par défaut qui recevra tout le trafic non attribué à une autre classe.

Les clauses **filter** indiquent quels paquets concernent quelles classes. On y trouve, dans l'ordre :

- l'interface.
- le nom de la classe.
- l'adresse IP source (0 équivaut à `0.0.0.0`, soit n'importe quelle adresse). On peut préciser un masque de sous-réseau avec le mot-clef **netmask**, ce qui permet de préciser des plages d'adresses ;
- le port source (0 équivaut à n'importe quel port).
- l'adresse IP de destination, accompagnée parfois d'un masque de sous-réseau ;
- le port de destination ;
- et le protocole de transport (6 correspond à TCP, conformément à ce qui est indiqué dans le fichier /etc/protocols).

Cet exemple n'illustre qu'une petite partie des possibilités d'ALTQ. Le lecteur curieux en acquerra la maîtrise en se référant à la documentation, à commencer par les pages **man** altq.conf et **altqd**.

Réseaux privés virtuels

À quoi servent les réseaux privés virtuels ?

Un pare-feu convient parfaitement pour isoler un réseau interne du reste de l'Internet. Correctement configuré, il permet le développement de tout un intranet sans qu'il faille se soucier de l'authentification des usagers, car le pare-feu assure qu'ils viennent bien du réseau privé.

Mais tôt ou tard, vous voudrez rendre votre intranet accessible à des usagers situés à l'extérieur du pare-feu. Il existe principalement trois scénarios : l'interconnexion de réseaux privés via Internet, l'accès distant des usagers itinérants à l'intranet (on parle alors souvent d'extranet), et la sécurisation des réseaux radio. Dans les

PIÈGE À C... **Pas d'authentification sur l'intranet**

En réalité, il est assez imprudent de considérer le réseau interne comme forcément inaccessible depuis l'extérieur. Même supposant le pare-feu exempt de trous de sécurité et d'erreurs de configuration, il reste un bon nombre de situations où cette supposition est incorrecte.
C'est notamment le cas des serveurs accessibles de l'extérieur mais qui ont été piratés et qui permettent de rebondir à l'intérieur, et celui des postes clients infestés par des logiciel espions (*spywares*), des virus, ou des chevaux de Troie.
Vous pouvez donc considérer votre réseau interne comme plus sûr que l'Internet, sans toutefois l'estimer complètement à l'abri des regards indiscrets. Si vous devez l'utiliser pour diffuser des données confidentielles, adoptez une authentification forte (SSH, Web sécurisé SSL...). Et espérez que les clients qui accèdent à l'information ne seront pas eux-mêmes compromis.

trois cas, ces problèmes peuvent se résoudre en mettant en œuvre des réseaux privés virtuels (VPN : *Virtual Private Network*).

Interconnexion de réseaux privés

Premier scénario, votre entreprise est multi-sites, et vous souhaitez que chacun des sites ait accès à l'intranet. On peut résoudre ce problème au prix fort, en louant des liaisons spécialisées pour raccorder les sites. On peut aussi adopter des liaisons intermittentes, par exemple en RNIS. La première solution est assez onéreuse, la seconde le deviendra pour des gros trafics.

L'alternative, c'est le VPN. On peut de nos jours raccorder à peu de frais chacun des sites à l'Internet, par exemple via le câble ou l'ADSL. Grâce à un VPN, vous pourrez utiliser l'Internet pour relier vos réseaux internes de façon sûre (c'est l'aspect privé du VPN), et en donnant à vos usagers l'illusion d'un réseau interne unique (c'est le côté réseau virtuel du VPN). La figure 10.2 illustre cette situation.

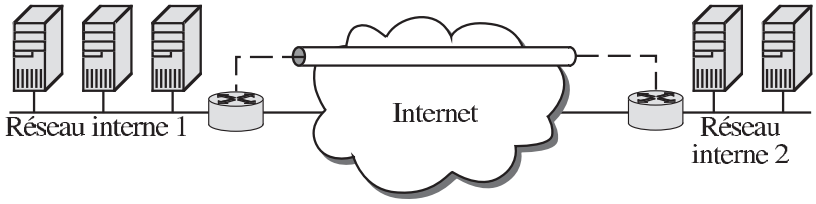


Figure 10–2 VPN reliant des réseaux locaux

CULTURE **Liaisons spécialisées et réseaux virtuels**

Certains opérateurs proposent un service de pontage entre les réseaux locaux de part et d'autre de la liaison spécialisée, donnant ainsi l'illusion d'un réseau virtuel où toutes les machines seraient placées sur le même réseau Ethernet. Bien entendu, ce service est vendu plus cher qu'une liaison spécialisée normale.

L'accès distant à l'intranet : l'extranet

Autre situation où le besoin en VPN se fait sentir : l'accès distant à un intranet. Vous avez des usagers itinérants, amenés à utiliser les moyens informatiques de l'intranet depuis l'extérieur du site. Ouvrir votre intranet à l'ensemble de l'Internet n'est pas envisageable, pour plusieurs raisons.

D'abord, tout système informatique comporte toujours quelques applications indispensables développées en dépit du bon sens, et incapables d'utiliser une authentification forte. Vous ne voulez pas que des informations confidentielles se retrouvent trivialement à la portée du premier indiscret venu. Quand bien même toutes vos applications seraient développées dans un souci de sécurité, ouvrir plus de services à l'extérieur signifie plus de services à surveiller, ce que vous n'avez peut-être pas les moyens de faire.

Une solution simple à mettre en œuvre, consiste à obliger les usagers distants à passer par une connexion sur une ligne téléphonique RTC ou RNIS, via un lien modem. Faire des écoutes sur le réseau téléphonique peut paraître plus difficile que sur Internet. C'est le cas dans une certaine mesure, tout étant évidemment affaire de moyens. Les principaux inconvénients de cette méthode sont la faible vitesse de la connexion (limitée à 56 kb/s en RTC ou 64 kb/s pour une ligne RNIS), et bien sûr le coût des communications.

Ici encore, le réseau privé virtuel apporte une solution satisfaisante. Sous réserve d'installer sur chaque client les logiciels adéquats, on peut lui donner un accès authentifié au réseau interne via l'Internet. L'expérience de l'usager sera semblable à celle d'une connexion par modem, à la vitesse près. La figure 10.3 illustre cette configuration.

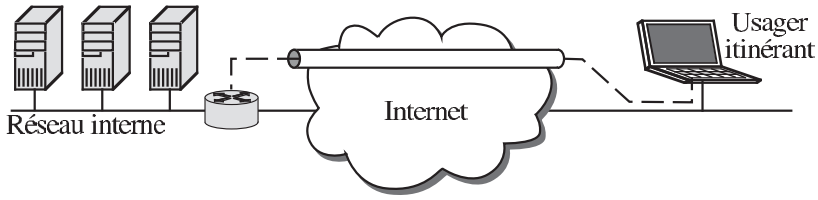


Figure 10–3 VPN Pour l'accès distant

Sécuriser les liaisons radio

VOCABULAIRE **Réseaux sans fils**

On parle d'Ethernet sans fil, de réseaux Wifi, pour *WIreless FIdelity* (fidélité sans fil), de WLAN, pour *Wireless Local Area Network* (réseau local sans fil), ou plus techniquement de réseaux 802.11, du nom de la norme IEEE qui les standardise.

Les réseaux Ethernet sans fils connaissent un franc succès. Qui refuserait de se débarrasser de ces encombrants câbles réseau, jamais de la bonne longueur ? Personne ? Personne, hélas, excepté celui qui ne souhaite pas fournir à son voisin un accès à ses moyens informatiques.

La sécurité des WLAN est assurée par un protocole nommé WEP (pour *Wireless Equivalent Privacy* : confidentialité équivalente au réseau filaire). Ce protocole, mal conçu dès l'origine, est criblé de problèmes de sécurité. Il existe des alternatives à WEP, mais les utiliser vous mènera à des problèmes d'interopérabilité, tous les équipements n'étant pas susceptibles de pratiquer d'autres protocoles.

Une solution pour utiliser les réseaux sans fils en toute sérénité est de repousser les problèmes de sécurité dans des couches réseaux plus hautes. Considérons le réseau sans fils comme fondamentalement non sûr et utilisons-le comme on utiliserait un lien à travers Internet. Pour cela, on fera du réseau sans fil un sous-réseau isolé, interdit de communication tant vers le réseau interne que vers l'Internet. L'unique moyen de sortir sera de passer par un VPN, qui seul transmettra les communications d'une machine sans fil au réseau interne, et réciproquement. On est ainsi ramené au problème de l'accès distant des usagers itinérants à l'intranet.

Fonctionnement du VPN

Le VPN fonctionne sur le principe du tunnel : chaque morceau du réseau privé est relié à l'extérieur par une passerelle VPN. Celle-ci va encapsuler les paquets

à destination des autres portions du réseau privé, et utiliser le lien non sûr mis à sa disposition pour les transmettre à la passerelle VPN tête de pont sur la portion de réseau privé destination. Cette dernière décapsule alors les paquets et les transmet à la machine concernée.

Dans le cas de l'accès distant à l'intranet depuis des postes d'utilisateurs itinérants, l'une des portions du réseau privé est réduite à une machine : le poste de l'utilisateur. C'est aussi cette machine qui joue le rôle de passerelle VPN. On appelle souvent la passerelle VPN située du côté du réseau interne serveur d'accès VPN ou concentrateur d'accès VPN,

Pour assurer la confidentialité des données, les passerelles VPN encapsulent les paquets dans un protocole capable d'authentifier, de chiffrer, et de garantir l'intégrité des données. Nous allons voir dans la section suivante quels protocoles sont disponibles.

Quelles solutions techniques pour les VPN ?

Le monde des VPN est une arène où bon nombre de protocoles s'affrontent, car l'IETF n'a pour le moment adopté aucun protocole en standard pour toutes les situations. Cet embarras du choix n'est pas la panacée en matière de protocoles.

Voici un aperçu des solutions les plus courantes :

- PPP sur SSH
- SSL
- PPTP
- L2TP
- IPsec
- L2TP sur IPsec

CULTURE **IETF**

L'IETF (pour *Internet Engineering Task Force*) est une organisation de concepteurs de réseaux, d'opérateurs, de fabricants de matériels et de chercheurs. Elle est ouverte à tout individu intéressé par l'évolution de l'Internet. Sa principale tâche est la mise au point des nouveaux standards, menée à travers la rédaction de RFC (*Request for Comments*). Le lecteur curieux en apprendra plus sur l'IETF en consultant son site Web : http://www.ietf.org.

PPP sur SSH

Cette solution consiste à encapsuler PPP dans SSH. Sur PPP, on refait passer de l'IP. Avantage de cette méthode : elle est simple et rapide à mettre en œuvre. Inconvénient : SSH fonctionnant sur TCP, faire passer des protocoles basés sur TCP dans le tunnel revient à passer deux fois par la case TCP. C'est désastreux sur le plan des performances, car les mécanismes de contrôle de flux des deux couches TCP se perturbent alors mutuellement.

De plus, l'infrastructure de clefs publiques (PKI : *Public Key Infrastructure*, voir l'encadré sur ce sujet) n'est pas intégrée dans cette solution. L'administrateur doit se soucier du déploiement des clefs publiques. Cette solution n'est donc praticable que pour monter des VPN reliant des réseaux, et si les performances ne sont pas importantes. On peut l'utiliser pour un accès distant des utilisateurs itinérants, mais elle reste réservée à des usagers disposant de compétences UNIX.

Voyons la mise en pratique. La figure 10.4 décrit la topologie d'un réseau comportant un VPN. On y trouve deux réseaux privés, sur les plages `10.0.2.0/24` et `10.0.4.0/24`. Chacun de ces deux réseaux est connecté à Internet par une

passerelle faisant office de translateur d'adresses. Entre ces deux passerelles, qui s'appellent Oreste et Pylade, on trouve un tunnel en PPP sur SSH.

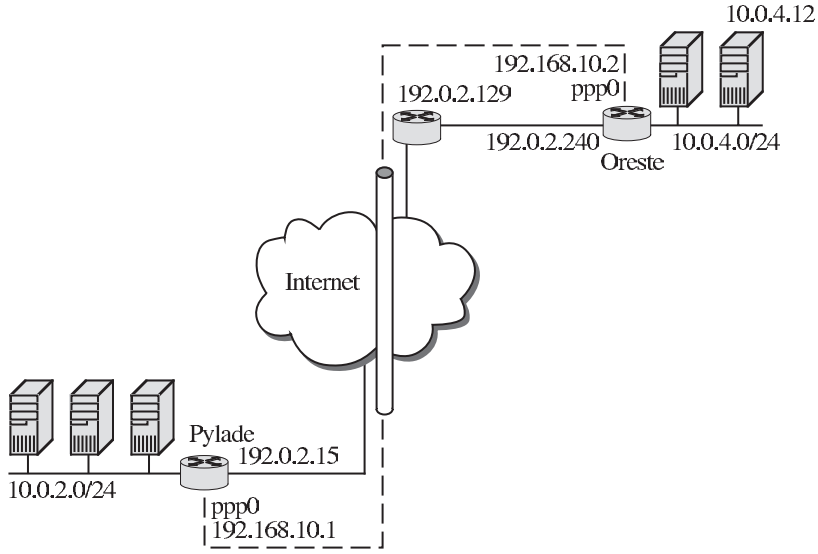


Figure 10–4 Topologie d'un réseau avec VPN

La situation est symétrique pour la communication entre les deux réseaux internes. En revanche, Pylade est la passerelle permettant de sortir pour le trafic en provenance des deux réseaux. Oreste a pour passerelle par défaut `192.0.2.129`. Nous ne nous intéresserons pas à la passerelle par défaut de Pylade.

SUR LES AUTRES UNIX **La commande `pppd`**

Cet exemple ne fonctionnera qu'avec une commande **`pppd`** assez récente pour connaître l'option **`pty`**. À l'heure où sont écrites ces lignes, c'est le cas sur NetBSD et la plupart des distributions GNU/Linux, mais pas sur FreeBSD et OpenBSD. Pour ces deux systèmes, il faudra se tourner vers la commande **`ppp`** pour obtenir une fonctionnalité similaire – à moins d'une éventuelle mise à jour de **`pppd`** survenue après la sortie de cet ouvrage.

```
oreste# ifconfig ppp0 ❶
ppp0: flags=8010<POINTOPOINT,MULTICAST> mtu 1500
oreste# /usr/sbin/pppd noauth pty "ssh root@192.0.2.15 \
> '/usr/sbin/pppd notty noauth 192.168.10.1:192.168.10.2'" \
> 192.168.10.2:192.168.10.1 ❷
oreste# ifconfig ppp0
ppp0: flags=8010<UP,POINTOPOINT,MULTICAST> mtu 1500
        inet 192.168.10.2 -> 192.168.10.1 netmask 0xffffffff ❸
oreste# ping -nc 1 192.168.10.1 ❹
PING 192.168.10.1 (192.168.10.1): 56 data bytes
64 bytes from 192.168.10.1: icmp_seq=0 ttl=243 time=294.478 ms

----192.168.10.1 PING Statistics----
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 294.478/294.478/294.478/0.000 ms
oreste# route add -host 192.0.2.15 192.0.2.129 ❺
add host 192.0.2.15: gateway 192.0.2.129
oreste# route delete default
delete net default
oreste# route add default 192.168.10.1 ❻
add net default: gateway 192.168.10.1

pylade# route add -net 10.0.4.0 -netmask 255.255.255.0 192.168.10.2 ❼
add net 10.0.4.0: gateway 192.168.10.2
pylade# traceroute -n 10.0.4.12 ❽
traceroute to 10.0.4.12 (10.0.4.12), 64 hops max, 40 byte packets
 1  192.168.10.2  235.780 ms  247.637 ms  281.198 ms
 2  10.0.4.12  234.581 ms  225.227 ms  230.911 ms
```

❶ Le tunnel PPP utilisera l'interface `ppp0`. Avant d'aller plus loin, vérifiez que cette interface n'est pas déjà utilisée – par exemple par un *daemon* **pppd** démarré lors d'une tentative précédente.

❷ Cette incantation va créer le tunnel PPP à travers SSH. Dans le cas présent, l'authentification se faisant par clef RSA, aucun mot de passe n'est demandé. Sans une telle clef, **ssh** demanderait un mot de passe.

❸ Une fois le tunnel créé, l'interface `ppp0` doit être marquée active (`UP`), et porter son adresse IP et celle de l'autre extrémité du lien PPP, ici `192.168.10.2` et `192.168.10.1`.

Si cela ne fonctionne pas, contrôlez /var/log/messages : ce fichier devrait contenir des messages de **pppd**. Pour reprendre à zéro, si cela est nécessaire, tuez **pppd** et supprimez les adresses allouées sur l'interface `ppp0` en tapant **ifconfig ppp0 delete 192.168.10.2 down**

❹ Autre manière de vérifier le bon fonctionnement du tunnel : un **ping** vers l'autre extrémité (ici Pylade).

❺ Notre but est maintenant d'envoyer tout le trafic issu de `10.0.4.0/24` vers Pylade. On ne peut pas se contenter de retenir `192.168.10.1` comme route par défaut, car le tunnel PPP sur SSH fonctionne par une communication entre `192.0.2.240` et `192.0.2.15`. On commence donc par enseigner à Oreste une route statique vers `192.0.2.15`, en passant par ce qui était initialement sa passerelle par défaut : `192.0.2.129`.

❻ Nous pouvons maintenant supprimer et remplacer la route par défaut sans perdre la connectivité entre Oreste et Pylade. La nouvelle route par défaut est `192.168.10.1`, c'est-à-dire Pylade à travers le tunnel PPP sur SSH.

❼ Sur Pylade, on installe la route inverse. La situation est ici nettement plus simple, puisque Pylade doit juste apprendre à atteindre `10.0.4.0/24` en passant par le tunnel. Aucune modification de la route par défaut n'est nécessaire.

❽ Avec **traceroute** ou **ping**, on peut vérifier le bon fonctionnement du VPN. À ce stade, un échec de ce test peut être dû à une mauvaise route – utilisez **tcpdump** de chaque côté pour voir quelle machine a un problème de routage. Autre possibilité : une règle de filtrage mal placée empêche **ping** ou **traceroute** de fonctionner.

Un piège à c... à vérifier éventuellement : un des routeurs ne route pas (ce qui se corrige sur BSD d'un coup de **sysctl -w net.inet.ip.forwarding=1**). Si vous n'arrivez pas à communiquer depuis l'un des réseaux privés vers l'extérieur, pensez aussi à vérifier d'éventuelles tables de translations d'adresses : dans l'exemple présent, `10.0.2.0/24` et `10.0.4.0/24` doivent tous deux être translatés sur Pylade.

ASTUCE **PPP sur SSH pour l'accès distant**

Si vous utilisez PPP via SSH pour l'accès distant, la machine nommée Pylade dans cet exemple devient le concentrateur d'accès, et Oreste la machine de l'usager itinérant. Toutes les commandes présentées restes valables, sauf l'installation de la route inverse de Pylade vers `10.0.4.0/24`, qui devient inutile, Oreste ne desservant plus aucun réseau.

Vous voudrez sans doute éviter de devoir invoquer une session SSH en tant que root vers le concentrateur d'accès. C'est possible, avec l'option **call** de **pppd**, qui permet à un utilisateur non privilégié de démarrer des **pppd** avec des options prédéfinies par un fichier placé dans /etc/ppp/peers. Consultez la page **man** de **pppd** pour plus de détails.

CULTURE **Le choix des adresses IP**

192.168.10.1 et 192.168.10.2 sont ici les adresses IP des interfaces côté tunnel des passerelles VPN. N'importe quelle adresse fait ici l'affaire, mais le trafic étant destiné à rester privé, on préfère souvent retenir des adresses non routables, sur les plages 192.168.0.0/16, 172.16.0.0/12, ou 10.0.0.0/8. Bien entendu, ces adresses ne doivent pas déjà être utilisées sur votre intranet.

PLUS LOIN **Le problème de l'authentification dans les VPN : l'importance de la PKI**

La conception d'une authentification sûre à distance sûre est un problème complexe, dont les détails dépassent largement les objectifs de ce livre. Livrons-nous donc simplement à un examen rapide du sujet.

L'authentification locale à un système UNIX consiste à taper un mot de passe, chiffré par une fonction à sens unique (fonction de hachage, telle que DES ou MD5), puis comparé à un enregistrement chiffré (*hash*) du mot de passe correct. Cette solution, simple, n'est sûre que parce que personne ne peut observer les flux d'informations entre votre clavier et votre unité centrale (si vous employez un clavier sans fil sans jamais vous être posé de questions de sécurité, c'est le moment de le faire...).

Pour une authentification à travers un lien non sûr, où des espions sont susceptibles d'intercepter voire d'altérer le trafic, le problème est plus complexe à résoudre.

Le niveau zéro de la sécurité, c'est le mot de passe en clair. C'est ainsi que fonctionne l'authentification PAP (*Password Access Protocol*) de PPP, ainsi que les authentifications Telnet ou FTP classiques. Cette transaction est décrite dans la figure 10.5.

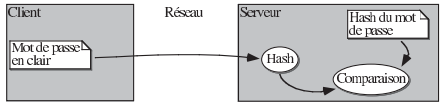


Figure 10–5 Authentification PAP

Le niveau juste supérieur consiste à envoyer à travers le réseau un hachage du mot de passe, qu'il faudra comparer au hachage attendu. Cette approche ne semble efficace qu'à première vue : ce hachage peut être capturé puis cassé en essayant de passer tous les mots de passe possibles. De plus, un assaillant peut aussi fabriquer un client qui enverra directement un hachage précédemment capturé, et ainsi s'authentifier sans même connaître le mot de passe. C'est l'attaque de type répétition de la transaction (*replay*).

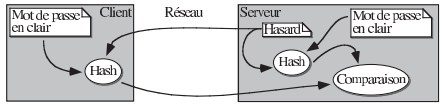


Figure 10–6 Authentification CHAP

Pour se protéger de telles manœuvres, on utilise des authentifications à question/réponse. Le client et le serveur partagent un secret ainsi qu'une fonction de hachage qui permet de calculer un résultat à partir du secret partagé et d'une donnée tirée au hasard. Le serveur envoie la donnée tirée au hasard au client : s'il est capable de renvoyer le hachage attendu, c'est qu'il connaît le secret partagé. La transaction est décrite dans la figure 10.6. Cette méthode évite la transmission du mot de passe en clair ou sous forme de hachage et il n'est pas possible de répéter la transaction puisque les valeurs sont tirées au hasard. L'authentification CHAP (*Challenge Response Authentication protocol*) de PPP utilise cette méthode. Son inconvénient est d'obliger à stocker le mot de passe en clair sur le serveur, ce qui peut être indésirable.

Microsoft a mis au point MS-CHAP, un CHAP propriétaire résolvant ce dernier problème, puis MS-CHAP version 2, qui impose des hachages plus robustes (hachage natif de Windows NT au lieu des anciens hachages du LAN Manager). Mais on a beau améliorer les authentifications par secret partagé, toutes souffrent d'un défaut impossible à corriger : comment s'accorder sur un secret partagé au travers de moyens de communication que l'on suppose non sûrs ? Le chiffrement efficace des communications ayant pour préalable l'authentification des parties, on ne peut y avoir recours pour résoudre ce problème : à quoi bon chiffrer si vous n'êtes pas sûr que celui qui déchiffre est bien celui que vous croyez ?

Pour résoudre ce problème, on utilise des authentifications à clefs asymétriques. Une clef publique permet d'encoder des données, que seule la clef privée est capable de décoder. La clef publique du client est connue du serveur. Pour authentifier le client, le serveur encode des données prises au hasard avec sa clef publique et les lui envoie. Si le client peut renvoyer les données déchiffrées, c'est qu'il dispose de la clef privée ; il est donc authentifié. La transaction est décrite dans la figure 10.7. Pour une bonne sécurité, l'authentification doit être double, chacun s'authentifiant auprès de l'autre. Une fois les identités des machines assurées, on peut échanger du trafic chiffré en étant sûr de la sécurité des communications. Une authentification au niveau utilisateur peut avoir lieu à travers ce canal sécurisé, par exemple par mot de passe. SSH et SSL fonctionnent sur ce principe.

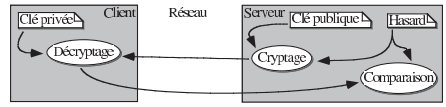


Figure 10–7 Authentification à clefs asymétriques

Mais comment échanger les clefs publiques en étant assuré de leur authenticité ? Pour SSH, le client garde la trace des clefs des serveurs qu'il a contactés. Si, lors de la prise de contact avec un serveur, il reçoit une clef publique inconnue, il en avertit l'utilisateur : soit l'administrateur du serveur a changé les clefs, soit un intrus s'est placé entre le client et le serveur, qui tente une attaque de type espion au milieu (*man in the middle*, ou MiM). Reste le problème de ce type d'attaques lors de la première connexion à un serveur, dont la véritable clef publique est inconnue.

On peut le résoudre en recourant à une infrastructure de clef publique (PKI : *Public Key Infrastructure*), qui permet de déployer des clefs publiques en assurant leur authenticité. C'est ce que fait SSL avec les certificats SSL à la norme x509. Le certificat est alors signé par la clef privée d'une autorité de certification, dont la clef publique est connue de tous.

SSL

Les VPN basés sur SSL désignent deux sortes de produits. D'une part, de vrais VPN, où SSL est utilisé comme un tunnel dans lequel on fait passer un protocole de réseau. OpenVPN est un produit Open Source assurant cette fonction. Il permet de faire du SSL sur UDP, ce qui coupe court aux problèmes de performances évoqués pour PPP sur SSH.

Simple à mettre en œuvre pour un tunnel entre deux réseaux, il peut aussi résoudre les problèmes d'accès distant des utilisateurs. Le seul obstacle dans ce cas est encore l'absence de client à interface graphique pour les systèmes d'exploitation « grand public » comme Windows ou MacOS. Encore une fois, cette solution sera donc réservée à des usagers capables de taper des lignes de commande.

On parle aussi de VPN SSL, ou de WebVPN, pour des solutions consistant à passer le trafic Web dans des connexions SSL. Les données traversent l'Internet protégées par SSL du client au serveur d'accès VPN. Elles peuvent ensuite finir leur chemin en clair sur le réseau interne.

Le WebVPN ne met pas vraiment en place un réseau virtuel. Il se contente d'assurer une centralisation de l'authentification via SSL sur le serveur d'accès VPN, pour toutes les applications Web, qu'elles soient ou non prévues pour une authentification. Il est donc impraticable pour relier des réseaux entre eux et ne peut être utilisé que pour l'accès distant aux applications Web, et seulement aux applications Web.

Son grand avantage est de ne pas nécessiter d'application particulière, n'importe quel navigateur Web sachant parler SSL pouvant servir de client. La PKI est intégrée par le biais du navigateur ; la mise en œuvre sur les postes clients sera donc très simple. Ce point en fait l'outil de choix pour l'accès distant aux applications Web de l'intranet.

ALTERNATIVE **Stunnel**

Les moyens alternatifs de mettre en place des tunnels dans SSL ne manquent pas. Stunnel permet par exemple de créer un tunnel d'un protocole basé sur TCP à travers SSL. Il est en général utilisé pour ajouter une fonctionnalité SSL à un service qui n'est pas prévu pour cela.
En faisant passer du PPP dans le tunnel SSL, on peut réaliser un VPN SSL. Cette solution utilisant deux fois TCP, elle est aussi mauvaise que PPP via SSH sur le plan des performances. On y gagne en revanche la possibilité d'utiliser une PKI.

VOCABULAIRE **VPN SSL et WebVPN**

Afin d'éviter de confondre ces deux types de produits basés sur SSL, on adoptera dans ce chapitre la distinction suivante : les véritables VPN, qui mettent en œuvre un réseau virtuel, seront appelés « VPN SSL ». Les solutions d'accès distant aux applications Web via SSL seront appelées « WebVPN ».
Le discours commercial des vendeurs de solutions d'accès distant ne fait pas du tout cette distinction.

PPTP et L2TP

PPTP (*Point-to-Point Tunneling Protocol*) et L2TP (*Layer 2 Tunneling protocol*) sont deux protocoles assez semblables. Tous deux encapsulent une ou plusieurs sessions PPP, ce qui leur permet de transporter plusieurs protocoles de niveau 3 – par exemple IP, NetBEUI et AppleTalk.

PPTP et L2TP sont des initiatives poussées toutes deux par des alliances de vendeurs de matériel ou de logiciel. Toutefois, PPTP est surtout dominé par Microsoft, qui en est à l'origine.

Côté sécurité, c'est le désastre. PPTP propose de faire du chiffrement, mais le protocole de Microsoft est tellement miné de problèmes de sécurité qu'il crée plutôt des réseaux où c'est l'aspect privé qui est virtuel. Si vous vous sentez concerné par la sécurité de votre réseau, ne faites pas confiance à PPTP pour la préserver.

CULTURE **L2TP et L2F**

L2TP est une évolution du protocole L2F (*Layer 2 Forwarding*) mis au point par Cisco. Une compatibilité ascendante est d'ailleurs assurée avec ce protocole.

L2TP, pour sa part n'assure aucun chiffrement. Il rejette cette responsabilité sur les couches de niveau inférieur et se contente de mettre en place des réseaux virtuels. Pour le côté privé, il faut avoir recours à L2TP sur IPsec.

Côté performances, PPTP et L2TP savent fonctionner sur UDP, ce qui en fait de bonnes solutions. En revanche, l'encapsulation supplémentaire imposée par le support multi-protocoles apparaîtra comme un surcoût inutile pour ceux qui souhaitent seulement transporter du trafic IP : pourquoi faire du PPP sur L2TP sur IPsec là où l'on peut se contenter de faire de l'IPsec en mode tunnel ?

Pour l'aspect pratique, il existe plusieurs clients et serveurs PPTP et L2TP. Pour mettre en œuvre un VPN avec l'un de ces deux protocoles, il faudra installer un *daemon* PPTP ou L2TP et le configurer de façon adéquate. Des exemples d'implémentations pour L2TP sont donnés à la fin de ce chapitre. Pour PPTP, il existe un client et un serveur initialement conçus pour GNU/Linux, mais qui peuvent être utilisés sur les systèmes BSD :

- PPTP client : http://pptpclient.sourceforge.net/
- PopToP, un serveur PPTP : http://www.poptop.org/

CULTURE **PPTP et l'ADSL**

PPTP est utilisé pour assurer l'authentification des utilisateurs sur l'offre ADSL de certains opérateurs. D'autres utilisent PPPoE.
PPTP client étant indispensable pour connecter une machine UNIX à Internet avec ces offres ADSL, ce logiciel a été très largement éprouvé.

IPsec

IPsec est un ensemble de deux options IP assurant l'authentification et la confidentialité des paquets, directement au niveau 3. On peut l'utiliser pour un trafic normal entre machines (mode transport), ou pour mettre en place des tunnels (mode tunnel).

Comme il opère au niveau 3, IPsec permet de sécuriser n'importe quel type de trafic IP sans imposer d'encapsulation supplémentaire. En revanche, pour les trafics non IP, il faut avoir recours à une encapsulation dans d'autres protocoles – comme L2TP.

IPsec n'a pas que des avantages. Ses implémentations étant relativement jeunes, on peut buter sur des problèmes d'interopérabilité entre solutions d'origines différentes.

De plus, les authentifications disponibles ne se plient pas à tous les usages. Elles se limitent à un niveau machine, avec soit un secret partagé, soit des certificats x509. C'est adéquat pour résoudre les problèmes d'interconnexion de réseaux privés,

SÉCURITÉ **PPTP ou l'insécurité informatique**

La version 1 de PPTP contenait un nombre de trous de sécurité effrayant pour un produit orienté sécurité. Même une fois connu, ce fait n'a pas empêché des armées de consultants de recommander ce système pour monter des VPN d'entreprise, ce qui ne manque pas de laisser rêveur.
Microsoft a corrigé un bon nombre de problèmes avec PPTP version 2, mais le protocole reste vulnérable à quelques attaques. En particulier, si un assaillant peut capturer les paquets d'une session PPTP, il pourra en déduire les mots de passe en cassant le hachage ; ce n'est qu'une question de temps.
PPTP n'est donc pas recommandé dans les situations où la sécurité importe réellement. Microsoft se tourne d'ailleurs vers L2TP sur IPsec dans les versions de Windows les plus récentes.

mais pas pour l'accès distant des usagers, puisqu'il n'y a pas d'authentification au niveau utilisateur.

Pour l'aspect pratique, la configuration d'IPsec n'est pas très simple. Il faut éventuellement recompiler un noyau avec les options adéquates, puis installer et configurer le *daemon* d'échange de clefs : pour NetBSD et FreeBSD, on utilise **`racoon`**, disponible dans les systèmes de paquetages. OpenBSD utilise pour sa part **`isakmpd`**, intégré au système de base.

Le lecteur intéressé est invité à consulter les documentations des projets BSD sur le sujet. À l'heure où sont écrites ces lignes, c'est le projet NetBSD qui fournit la documentation la plus complète (elle s'applique aussi en grande partie à FreeBSD).

Foire Aux Questions IPsec du projet NetBSD : http://www.netbsd.org/fr/Documentation/network/ipsec

Section sur les VPN IPsec du *handbook* FreeBSD : http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/ipsec.html

L2TP sur IPsec

L2TP et IPsec semblent des solutions complémentaires. L2TP gère l'encapsulation multi-protocoles et l'authentification niveau utilisateur, alors que IPsec prend en charge un chiffrement robuste. Du point de vue de la sécurité comme de celui des performances, cette solution est satisfaisante.

Pour mettre en œuvre un VPN en L2TP sur IPsec, la première étape est la configuration d'IPsec, dont les grandes lignes sont évoquées dans la section précédente. La deuxième partie consiste à faire fonctionner un *daemon* L2TP. Il existe un certain nombre d'implémentations du protocole L2TP en logiciel libre, aucune n'étant à ce jour très mûre. Deux implémentations couramment utilisées :

- **`l2tpd`** : http://www.l2tpd.org
- *Roaring Penguin* L2TP : http://sourceforge.net/projects/rp-l2tp/

PLUS LOIN **Authentification utilisateur et IPsec**

Le premier réflexe qui vient à l'esprit pour une authentification niveau utilisateur, c'est le très classique couple nom d'utilisateur et mot de passe. IPsec permet d'utiliser un secret partagé pour l'authentification, mais pas de nom d'utilisateur. Chaque machine est identifiée par le couple adresse IP et secret partagé. Si les usagers se connectent depuis des adresses dynamiques, on devra donc accepter tous les secrets partagés possibles de toutes les adresses possibles, ou prendre le même secret partagé pour tous les usagers. Dans les deux cas, la solution n'est pas très satisfaisante sur le plan de la sécurité.

L'alternative, ce sont les certificats x509, qui vont assurer une bonne authentification de la machine. Le problème de l'authentification utilisateur est repoussé aux couches supérieures, à moins de distribuer des certificats x509 personnels aux usagers.

Mais que choisir ?

Aucune de ces solutions n'est parfaite. Le tableau 10.1 récapitule les mérites et inconvénients de chacune d'entre elles.

Dans le choix d'une solution VPN, tout dépend de l'application envisagée. Pour relier entre eux des réseaux privés, les VPN SSL et IPsec semblent les meilleures solutions. Les VPN SSL seront probablement plus simples à mettre en œuvre, alors que les VPN IPsec seront meilleurs en terme de performances.

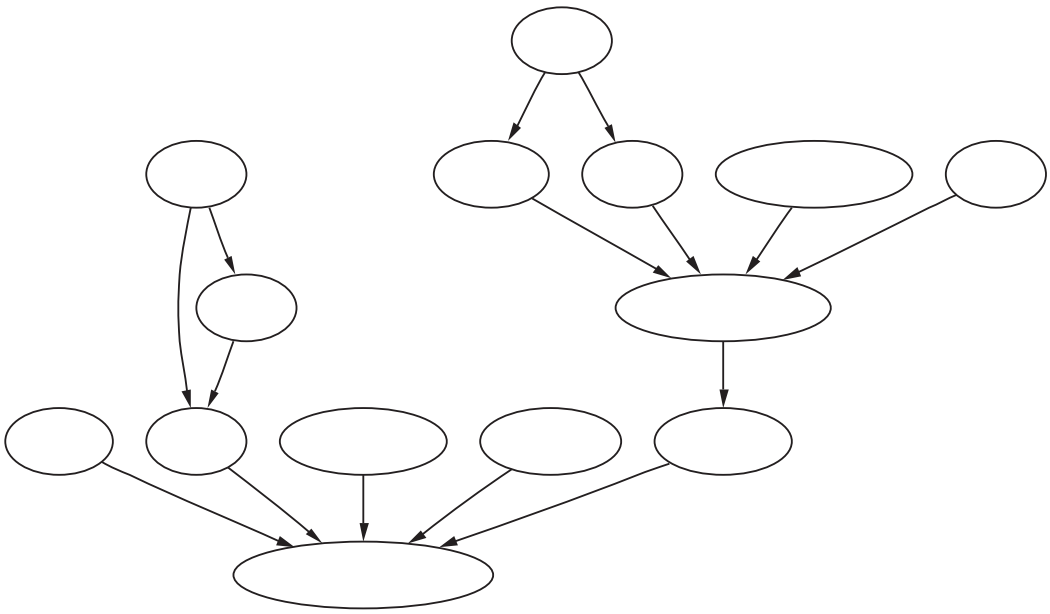
Pour l'accès distant, la condition nécessaire au déploiement d'une solution est souvent la disponibilité de clients conviviaux pour les systèmes grands publics. Windows propose un client PPTP depuis NT4, et L2TP sur IPsec depuis Windows 2000. MacOS propose un client capable de PPTP et L2TP sur IPsec depuis la version X.3 (dite *Panther*).

Pour prendre en charge une gamme plus large de systèmes, on peut se tourner vers les vendeurs de matériel. Cisco propose par exemple des serveurs d'accès VPN (gamme Cisco 3000) IPsec et WebVPN, distribués avec une licence illimitée du logiciel de client VPN Cisco. Celui-ci fonctionne sur un grand nombre de versions des systèmes d'exploitation grand public, y compris les plus anciennes.

Solution	Avantages	Inconvénients
PPP/SSH	· Rapide à mettre en œuvre · Outils présents dans les distributions de base des BSD · Bonne sécurité	· Performances médiocres · Pas de client convivial pour les systèmes grand public
VPN SSL	· Bonnes performances · Possibilité d'utiliser une PKI · Bonne sécurité	· Pas d'interopérabilité entre solutions différentes · Pas de client convivial pour les systèmes grand public · Limité à IP
WebVPN	· Bonnes performances · Un navigateur suffit comme client · Intégration de la PKI · Bonne sécurité	· Limité aux applications Web
PPTP	· Multi-protocoles · Disponible de base dans Windows depuis NT4	· Non sûr · Encapsulation inutile pour les VPN en tout IP
L2TP	· Multi-protocoles	· Non sûr · Encapsulation inutile pour les VPN en tout IP
IPsec	· Bonne sécurité · Bonnes performances · Possibilité d'utiliser une PKI	· Problèmes d'interopérabilité · Pas d'authentification utilisateur · Limité à IP
L2TP/IPsec	· Bonne sécurité · Multi-protocoles · Possibilité d'utiliser une PKI	· Problèmes d'interopérabilité · Encapsulation inutile pour les VPN en tout IP

Tableau 10.1 Mérites et inconvénients des solutions pour les VPN

11



Logiciels tierce-partie et systèmes de paquetages

Un système d'exploitation n'a d'intérêt que dans la mesure où il propose des applications. Dans ce chapitre, nous aborderons l'installation de programmes tierce-partie développés pour le système. Nous verrons aussi qu'il est parfois possible, moyennant quelques efforts, d'utiliser des programmes absolument pas développés pour le système.

SOMMAIRE

MOTS-CLEFS

- Logiciels libres
- Compilation
- Paquetage
- Compatibilité binaire

Quels logiciels, et où les trouver ?

Même si le développeur ou l'administrateur peuvent se passionner pour des manipulations techniques du système, l'utilisateur final réclame des applications. Le système n'est à ses yeux, et à raison, qu'un socle édifié pour supporter d'autres programmes.

Il convient donc d'installer des applications. Les systèmes BSD comprennent déjà dans leurs distributions de base un bon nombre d'outils, déjà évoqués. Ils proposent clients et serveurs pour des protocoles de l'Internet : la messagerie avec Sendmail, le DNS avec BIND, SSH avec OpenSSH, FTP, Telnet, NTP. On y trouve aussi tous les outils nécessaires au démarrage en réseau (*netboot*) : DHCP, BootP, RARP, TFTP. Ainsi que le nécessaire pour construire des réseaux de machines Unix étroitement liées : NIS, NFS, Kerberos.

Tout cela ne suffira sans doute pas : il vous faudra des serveurs Web, des serveurs de fichiers ou d'impression pour machines Windows ou MacOS, des bases de données relationnelles fiables et performantes, qui pourront être connectées à un serveur Web pour servir du contenu dynamique ; des serveurs de *news* (groupes de discussion), de conversation en direct (*chat*) et bien d'autres choses encore : les besoins et envies évoluent sans cesse. Pourquoi pas des postes bureautiques avec navigateurs Web, traitements de texte, tableurs, logiciels de dessin en mode point (*bitmap*) ou vectoriel, logiciels de présentation, voire des mastodontes intégrant tout cela à la fois, suivant ainsi une formule qui a fait le succès de Microsoft ? La logithèque des logiciels libres comprend tout cela, et plus encore, dans ses milliers de titres.

Les logiciels libres peuvent être utilisés et distribués gratuitement. Oubliés les problèmes de comptage de licences, imposés à l'administrateur système par les éditeurs de logiciels. Mais toute médaille a son revers, et l'installation de ces programmes n'est pas toujours aussi facile que dans le cas de leurs équivalents propriétaires sous Windows ou MacOS.

EN PRATIQUE **Quelles applications passionneront vos utilisateurs ?**

Vous constaterez que quoi qu'ils en disent, les utilisateurs ne sont pas si difficiles : un économiseur d'écran bien choisi les laissera de nombreuses heures dans un état quasi-hypnotique.

Installation « à la main »

La plupart du temps, l'équipe de développement d'un logiciel libre fournit un code source à compiler pour le faire fonctionner. Il faut donc disposer d'outils de compilation, qu'on commencera par installer. Les trois systèmes BSD fournissent les outils de compilation dans le système de base, mais il est possible de ne pas les avoir sélectionnés lors de l'installation.

ALTERNATIVE **Les systèmes de paquetages**

Cette section aborde la compilation « à la main » des logiciels libres. La plupart du temps, on économise ses forces et recourt aux systèmes de paquetages (*packages*) ; il seront abordés dans la section suivante.

Vous pouvez très bien sauter cette section et n'y revenir que le jour où vous voudrez installer un logiciel depuis les sources sans passer par les systèmes de paquetages.

DORMEZ TRANQUILLE **Droits pour l'installation**

Nul besoin d'être root pour télécharger, décompacter, et compiler un logiciel. Ce n'est qu'au moment de l'installation proprement dite que les droits de root sont (parfois) nécessaires. Évitez autant que possible de travailler en tant que root : vous éviterez de fausses manœuvres funestes.

Pourquoi ne distribuent-ils pas de binaires ?

Les habitués du monde Windows ou MacOS s'étonneront de l'absence de distribution sous forme de binaires déjà compilés. L'explication est simple : les programmes conçus pour Unix tentent d'être portables, c'est-à-dire capables de

compiler et fonctionner sur plusieurs systèmes différents. Or, on trouve des dizaines de variétés d'Unix, dont certains sont proposés sur un nombre important de processeurs différents.

Pour servir tout le monde, il faudrait produire des centaines de binaires différents, que l'équipe de développement a rarement les moyens de produire et de tester. Certains projets distribuent des binaires aux formats les plus courants, tels que Win32/i386, Linux/i386, Solaris/sparc ou Darwin/PowerPC ; les autres plateformes devront recompiler. Les systèmes BSD, peu connus, sont rarement prévus dans les distributions binaires, si l'on excepte les systèmes de paquetages, analysés plus loin.

B.A.-BA **Quel binaire pour quel système ?**

Un programme est compilé pour une version du noyau du système d'exploitation et pour un processeur particulier. Si de plus il est lié dynamiquement, il sera également prévu pour une version donnée des bibliothèques (*libraries*) dont il dépend.
Certains systèmes d'exploitation maintiennent une compatibilité binaire entre les différentes versions, ce qui élimine la dépendance sur la version du système. Ne subsistent donc que la dépendance sur le système lui-même et sur le processeur.

Trouver et télécharger les sources

Les projets de logiciels libres sont très peu centralisés : il existe des milliers de développeurs isolés ou d'équipes de développement, travaillant sans se coordonner. Quelques sites permettent heureusement d'en recenser une bonne partie :

- http://www.gnu.org/directory/GNU/
- http://savannah.gnu.org/
- http://freshmeat.net/
- http://www.sourceforge.net/

En cherchant un peu, on finit donc toujours par trouver le point de distribution des sources d'un projet de logiciel libre. Cette recherche peut s'avérer fastidieuse, et il est clair qu'un système de paquetages, déjà informé de ces adresses, ne pourrait que faciliter les choses.

Pour le téléchargement proprement dit, certains outils manqueront peut-être à l'appel. Dans le cas des sites FTP, tous les Unix produits il y a moins de 10 ans proposeront la commande **`ftp`**. Dans le cas du protocole HTTP (c'est-à-dire sur le Web), on peut recourir à des commandes comme **`wget`**, **`w3m`**, **`links`**, ou **`lynx`**. Absentes des systèmes les plus anciens, elles ne sont pas toujours proposées par défaut dans les systèmes récents.

En particulier, ces commandes ne sont pas disponibles dans la distribution de base des systèmes BSD, mais ces derniers ne sont pas démunis pour autant. Leur commande **`ftp`** est capable de charger un fichier sur un site Web si on lui fournit une URL en http:// comme suit :

```
$ ftp http://ftp.easynet.be/samba/ftp/pre/samba-2.2.8pre2.tar.gz
Requesting http://ftp.easynet.be/samba/ftp/pre/samba-2.2.8pre2.tar.gz
 16% |*****                              |   832 KB   59.39 KB/s    01:13 ETA
```

PLUS LOIN **Comment compiler le compilateur ?**

Il est impossible de compiler le compilateur sans compilateur. Pour travailler sur un système sans compilateur disponible en version binaire (ce qui n'est jamais le cas sur les BSD), il faudra procéder à une compilation croisée (*cross compile* en anglais, on dit aussi souvent *cross compilation* en jargon français) du compilateur, c'est-à-dire le compiler depuis une autre plateforme, disposant elle d'un compilateur fonctionnel. C'est une procédure difficile ; consultez la documentation officielle avant de vous y risquer : http://gcc.gnu.org/install/build.html

CULTURE **Navigateurs en mode texte**

`w3m`, **`lynx`**, et **`links`** sont des navigateurs en mode texte. Ils surprendront les non habitués, mais permettront aux initiés de naviguer sur le Web dans une session texte, ce qui peut parfois s'avérer très pratique – il est même possible de voir les images dans certaines conditions ! Extrêmement économes en ressources, ces programmes se pilotent et contrôlent efficacement et rapidement.

Décompactage et gestion des dépendances

Les codes source sont en général distribués au format .tar (fichier unique permettant de rassembler une arborescence), compressé par les programmes **`gzip`**, **`compress`** ou **`bzip2`** respectivement en .tar.gz (ou .tgz), .tar.Z, ou .tar.bz2. On trouve parfois des archives en .zip ou .shar. Il faudra déballer tout cela, et parfois

installer les outils nécessaires à cette opération. Mais comment s'assurer qu'une archive ne contient pas un cheval de Troie ?

Votre arme contre les chevaux de Troie : la somme de contrôle

Après avoir téléchargé un fichier de sources, c'est une bonne habitude que d'en vérifier la somme de contrôle (*checksum*). Elle accompagne généralement l'archive sur son lieu de téléchargement, dans un fichier voisin. On peut calculer des sommes de contrôle avec les commandes **`cksum`**, **`md5`**, ou **`sha1`**. Exemple avec la commande **`md5`** :

```
$ md5 archive.tar
MD5 (archive.tar) = d590f78d014d13df952348bec97c48a2
```

La somme de contrôle permet de détecter qu'une archive a été modifiée. Ce problème n'a rien de théorique : des malveillants ont déjà réussi à s'introduire sur un serveur de distribution de logiciels libres et ont placé un cheval de Troie dans une archive. En 2002, des programmes comme OpenSSH, Sendmail ou **`tcpdump`** ont été compromis pendant quelques jours avant que l'on ne s'aperçoive de la supercherie, grâce justement aux sommes de contrôle.

Évidemment, un indélicat capable de modifier l'archive pourra aussi remplacer le fichier consignant sa somme de contrôle, s'il est placé sur la même machine. Une bonne solution consiste à contrôler une somme de contrôle stockée à un endroit différent du fichier original. Nous verrons plus loin comment les systèmes de paquetages des systèmes BSD prennent ce problème en charge.

Après contrôle de l'intégrité du fichier de code source, il est temps de passer à son décompactage, qui dépend du format de l'archive. On remarquera évidemment que les distributions d'un outil de décompactage ne se font jamais dans le format

CULTURE **Cheval de Troie**

Les virus sont rares et difficiles à concevoir sous Unix : les programmes sont en général exécutés consciemment (par opposition à des programmes exécutés automatiquement et de manière transparente lors par exemple de la lecture d'un message) et avec les droits des utilisateurs. Les programmes les plus sensibles sont ceux qui sont parfois exécutés avec les droits de root, car ils peuvent intervenir sur toutes les composantes du système.
Un cheval de Troie est un élément d'un programme qui se comporte de manière non documentée et généralement non désirée, avec pour but de fournir à son concepteur un accès à la machine ou plus simplement, de l'endommager. C'est une allusion au stratagème antique qui a permis de prendre la cité de Troie après un long siège : on accueille avec confiance en son sein un élément qui s'avérera funeste et causera notre perte.

CULTURE **Sommes de contrôle**

Les sommes de contrôle sont un moyen simple de contrôler l'intégrité d'un fichier : c'est une référence à une méthode simple qui consiste à additionner tous les octets ou groupes d'octets d'un fichier, modulo une valeur maximum (comme 2 puissance 32 pour une somme sur 32 bits). L'objectif est d'obtenir rapidement un résultat court qui variera significativement pour toute modification du fichier non désirée (comme une corruption lors d'un transfert, ou l'ajout d'un cheval de Troie). Évidemment, cette méthode classique, fort insuffisante, ne détectera pas l'interversion de deux octets ou groupes d'octets ; les méthodes plus récentes sont plus solides.
La clef du numéro INSEE en France est un exemple de mauvaise somme de contrôle. On l'obtient en divisant le nombre formé des treize premiers chiffres par 97 et en conservant le reste de cette opération, qu'on complémente ensuite à 97. Exemple : le numéro INSEE imaginaire 1234567890123 vaut 12727504021 fois 97 plus 86. Pour compléter à 97 il manque 11 ; ce numéro impossible aurait donc pour clef 11. Cette méthode est mauvaise : par exemple, une personne qui confond les deux derniers paquets de deux chiffres de son numéro avec clef mais qui connaît cet algorithme ne pourra pas nécessairement décider quel est le bon numéro. En réalité, la plupart de ces interversions forment des numéros valables : 1234567890111 a pour clef 23.
La somme de contrôle n'est pas propre au fichier : puisque dans la plupart des cas elle est plus courte que le fichier qu'elle représente, il existe de nombreux autres fichiers qui aboutiront au même résultat. La plupart ont un contenu incompréhensible, et l'objectif est qu'il soit fort peu probable de tomber par hasard sur la même somme de contrôle en modifiant le fichier, sciemment ou par hasard. C'est la longueur de la somme de contrôle et la qualité de l'algorithme qui assurent cela. **`md5`** travaille par exemple sur 16 octets, ce qui représente un nombre énorme.

produit par cet outil : si on souhaite le compiler et l'installer, c'est *a priori* qu'on n'en dispose pas !

Fichiers .zip

Ce sont des archives renfermant des arborescences de fichiers qu'on peut extraire avec la commande **unzip**, absente par défaut de la plupart des systèmes Unix, dont les BSD. Il faut donc commencer par compiler et installer le couple **zip** et **unzip** (respectivement, compacteur et décompacteur). Il est disponible à l'adresse suivante : http://www.info-zip.org/, aux formats .tar.gz ou .tar.Z.

Fichiers .tar.gz

Ce sont des archives **tar** compressées au format **gzip**. C'est la commande **gunzip**, présente par défaut sur les systèmes BSD et sous GNU/Linux, qui permet de les décompacter, produisant ainsi une archive au format .tar. Les Unix dépourvus de **gunzip** devront installer et compiler cette commande ; ses sources sont disponibles au format .tar à l'adresse ftp://ftp.gnu.org/gnu/gzip/.

On utilise parfois l'extension .tgz comme synonyme de .tar.gz. **tar** est souvent capable de travailler en compactant ou décompactant ce format à la volée ; voir plus loin comment.

Fichiers .tar.Z

Semblables aux fichiers .tar.gz, ils ont été compactés avec la commande **compress**, disponible sur la plupart des Unix constructeur, ainsi que son pendant, **uncompress**, pour le décompactage. Ces commandes sont présentes partout pour des raisons historiques. En désespoir de cause, on pourra traiter un fichier .Z avec **gunzip**, qu'il faudra éventuellement installer. **tar** est souvent capable de travailler en compactant ou décompactant ce format à la volée ; voir plus loin comment.

Fichiers .tar.bz2

Les fichiers .tar.bz2 sont des archives **tar** compressées au format **bzip2**, récent et plus efficace que **gzip**. C'est **bunzip2** qui permet de les décompacter. Présent sur les systèmes BSD et GNU/Linux, il pourra manquer à l'appel sur des Unix plus anciens. On trouvera alors son code source à l'adresse http://sources.redhat.com/bzip2/, au format .tar.gz. **tar** est souvent capable de travailler en compactant ou décompactant ce format à la volée ; voir plus loin comment.

PLUS LOIN **Signatures cryptographiques**

La meilleure réponse à ces problèmes de compromission des fichiers de code source consiste à les signer à l'aide d'une clef privée, la clef publique correspondante étant largement répandue. Cette méthode, encore peu exploitée, permettrait d'améliorer considérablement la sécurité… à condition bien sûr que l'utilisateur dispose des outils cryptographiques nécessaires pour contrôler la signature.

EN PRATIQUE **Format .zip**

Ce format .zip est le même que celui en vigueur sur les systèmes Windows, où on le traite avec pkZip (gratuit), ou WinZip (payant).

ALTERNATIVE **Extensions .bz et .bz2**

D'abord appelé **bzip** (avec extensions en .bz), cet algorithme, ce programme et ce format ont dû être retouchés par leur auteur pour de sombres histoires de brevets. La nouvelle version, **bzip2**, est un peu moins efficace mais un peu plus rapide. Elle est capable de décompacter les archives .bz mais n'en créera plus dans ce format. Les archives créées par **bzip2** utilisent l'extension .bz2.

CULTURE **Archives `tar`**

`tar` signifie *Tape ARchive*, ou archive sur bande. À l'origine utilisée pour les sauvegardes sur bandes, cette commande est aujourd'hui surtout employée pour distribuer des sources.

PERFORMANCES **Affichage verbeux**

L'affichage verbeux ralentit sensiblement l'exécution de **`tar`** : on le déconseillera aux gens pressés.

B.A.-BA **Écriture compacte des options**

`tar -xvf archive.tar` représente **`tar -x -v -f archive.tar`**. Ce type d'écriture compacte des options courtes (d'une lettre) est commun à de nombreuses commandes sous Unix.

PIÈGE À C… **Où faites-vous votre extraction ?**

Prenez bien garde à l'endroit où vous vous trouvez avant de procéder à une extraction avec **`tar`** ou **`unzip`**. Certaines archives s'extraient proprement dans un unique sous-répertoire, mais d'autres déballent des centaines de fichiers dans le répertoire courant. Les commandes **`tar -tf archive.tar`** et **`unzip -l archive.zip`** fourniront le contenu d'une archive sans l'extraire ; vous pourrez ainsi contrôler que tout se passe bien.

Si vous oubliez cette précaution, la ligne suivante pourrait vous tirer d'affaire, mais ne la maniez qu'avec précaution car elle est potentiellement dangereuse :

```
$ rm -Rf `tar -tf archive.tar`
```

EN PRATIQUE **Dépendances de compilation**

Aux programmes et bibliothèques indispensables à l'exécution, il faut ajouter ceux qui sont requis à la compilation, inutiles une fois le programme compilé.

Fichiers .tar

Les fichiers .tar sont de simples archives renfermant des arborescences de fichiers, concaténés bout à bout avec des informations sur leurs droits et leurs chemins relatifs. Ces archives sont souvent compactées par différents algorithmes pour fournir des fichiers suffixés en .tar.gz, .tar.bz2, ou .tar.Z.

Les archives .tar s'extraient avec la commande **`tar`**, dont le syntaxe varie un peu avec les systèmes. Sous BSD et GNU/Linux, la syntaxe est la même. Voici un exemple de commande pour extraire le contenu d'une archive :

```
$ tar -xvf archive.tar
```

La page **`man`** détaillera cela, mais nous pouvons expliquer brièvement : l'option **`-x`** demande d'« eXtraire » le contenu de l'archive désignée par l'argument de l'option **`-f`** (comme « Fichier »). **`-v`** demande un affichage « Verbeux », affichant la liste des fichiers extraits.

Les commandes **`tar`** des systèmes BSD et GNU/Linux disposent d'une option **`-z`** qui permet de décompacter directement des archives compressées en .gz ou .Z. On évite ainsi la procédure en deux étapes qui consiste à décompacter avant d'extraire l'archive .tar, et qui mobilise de l'espace disque pour stocker l'archive décompactée. Exemple d'usage de l'option **`-z`** :

```
$ tar -xzvf archive.tar.gz
```

Les commandes **`tar`** des systèmes BSD et GNU/Linux admettent aussi une option **`--use-compress-program`**, qui permet de préciser explicitement le programme externe avec lequel on désire décompacter.

Fichiers .shar

Les fichiers .shar sont des auto-extractibles écrits en shell (**`shar`** signifie *SHell ARchive*). Ils s'extraient simplement en exécutant le fichier d'archive. Exemple d'utilisation :

```
$ sh ./bootstrap.shar
x - bootstrap/Makefile
x - bootstrap/crt0.S
x - bootstrap/display.c
x - bootstrap/help.txt
x - bootstrap/lcrt0.S
x - bootstrap/txt2asm.perl
x - bootstrap/xcrt0.S
```

Quoi encore ? Les dépendances

Nous avons vu que la simple extraction des sources d'une archive pouvait impliquer de nombreuses dépendances : pour décompacter un fichier .zip, il faut installer **`unzip`**, et pour installer **`unzip`**, il faut **`tar`**, **`gunzip`** et un compilateur. Cela n'est rien à côté de ce qui nous attend maintenant, puisque le programme a souvent besoin pour fonctionner d'autres programmes ou de bibliothèques, pas toujours fournis avec le système d'exploitation.

Ces dépendances peuvent porter sur un programme unique, tel que Perl, ou être absolument monstrueuses : des applications comme Mozilla ou OpenOffice.org dépendent de dizaines de bibliothèques et de programmes. Indiquées dans les notes d'installation du logiciel, ces dépendances pourront être exhaustives ou partielles, l'auteur supposant le système suffisamment moderne pour disposer de certaines bibliothèques. Parfois, la documentation n'indiquant aucune dépendance, elles se manifesteront lors de divers messages d'erreur en chemin.

Où trouver les notes d'installation du logiciel ? Après extraction de l'archive du code, on obtient un répertoire comme suit (exemple des sources d'OpenJade, moteur de rendu pour les fichiers SGML) :

```
$ tar -xzf openjade-1.3.tar.gz
$ ls -l
total 2
-rw-r--r--  1  manu  wheel  1165688 Feb 23  2000 openjade-1.3.tar.gz
drwxr--r--  26 manu  wheel  1024    Feb 15 13:49 openjade-1.3
$ ls -F openjade-1.3
COPYING             NEWS                generic/            nsgmls/
FILES               README              grove/              pubtext/
Makefile            SP.dsw              groveoa/            sgmlnorm/
Makefile.comm       SP.mak              include/            sp-generate.mak
Makefile.comm.in    VERSION             jade/               spam/
Makefile.in         all/                jade-generate.mak   spent/
Makefile.lib        build-win32.bat*    jade.dsw            spgrove/
Makefile.lib.CC     config/             jade.mak            style/
Makefile.lib.in     configure*          jadedist/           sunfix.sh
Makefile.lib.sun    contrib/            jadedoc/            sx/
Makefile.prog       develdoc/           japan.sgmldecl      testsuite/
Makefile.prog.in    doc/                lib/                unicode/
Makefile.wat        dsssl/              msggen.pl
```

Les fichiers README et INSTALL et les contenus des répertoires documentation ou doc (présent dans l'exemple) sont autant de sources potentielles d'information à explorer avant de se lancer. On peut perdre beaucoup de temps à retrouver la solution d'un problème déjà mentionné dans la documentation.

Trêve de préliminaires, passons à la compilation proprement dite.

ATTENTION **Compilations à éviter**

Évitez de faire vos premières armes en compilation avec Mozilla ou OpenOffice.org. Gigantesques, ces logiciels pourront poser des problèmes nombreux et inextricables pour un débutant.

CULTURE **SGML**

SGML est l'ancêtre de XML. Pour être plus précis, XML en est une simplification. Ces « méta-langages » définissent tous deux une syntaxe de marquage des fichiers, sans proposer de sémantique. Seules leurs applications (comme HTML, SMIL ou SVG) donnent un sens aux balises.

Configuration et compilation

Les programmes conçus pour Unix ont des sources portables, capables de compiler sur la plupart des systèmes. Hélas, quelques inévitables petites variations imposent souvent l'utilisation d'options particulières pour le compilateur, ou l'inclusion d'un fichier d'en-têtes particulier. Sans cela, la compilation échouera.

Cette portabilité est donc subordonnée à une petite phase de configuration du système de compilation, dans la plupart des cas menée par un script shell. Souvent appelé **`configure`**, il portera un autre nom dans certaines archives (Sendmail utilise **`Build.sh`**, qui prépare puis effectue la compilation).

CULTURE **Normes POSIX et scripts de configuration**

L'IEEE produit les normes POSIX (*Portable Operating System Interface*, ou interface de système d'exploitation portable). POSIX.1 décrit les interfaces de programmation standard en langage C et POSIX.2 décrit l'environnement de commandes. Ces normes ne spécifient pas tout, mais POSIX.2 standardise suffisamment de comportements pour qu'on puisse écrire des scripts shell très portables, et capables de configurer l'environnement de compilation de façon adéquate.
Les spécifications utilisées par l'*Open Group* pour attribuer l'usage de la marque UNIX s'appuient sur les normes POSIX.

Oh non, encore des dépendances !

Le script **`configure`** vérifie également les dépendances du programme, et signale toute erreur ou carence à ce sujet. Lors de son exécution (on l'invoquera en tapant **`./configure`** si le répertoire courant, . (point), comme nous l'avons

recommandé, ne fait pas partie du PATH), il produit une longue liste de tests et leur résultats :

```
$ ./configure
creating cache ./config.cache
saving distribution makefiles...
checking for gcc... gcc
checking whether the C compiler (gcc  ) works... yes
checking whether the C compiler (gcc  ) is a cross-compiler... no
checking whether we are using GNU C... yes
checking whether gcc accepts -g... yes
checking for a BSD compatible install... /usr/bin/install -c
checking whether ln -s works... yes
checking for c++... c++
checking whether the C++ compiler (c++ -O2 ) works... yes
checking whether the C++ compiler (c++ -O2 ) is a cross-compiler... no
checking whether we are using GNU C++... yes
checking whether c++ accepts -g... yes
checking for perl... /usr/pkg/bin/perl
checking how to run the C preprocessor... gcc -E
checking for ANSI C header files... yes
checking for limits.h... yes
checking for working const... yes
checking for inline... inline
checking for size_t... yes
checking for st_blksize in struct stat... yes

(...)
```

PLUS LOIN **Créer un script `configure`**

Le contenu des scripts **`configure`** semble artificiel, à raison. À de rares exceptions près, ils sont créés par le programme **`autoconf`**, à partir d'une liste de tests spécifiée dans un fichier configure.in (ou configure.ac). Souvent distribué avec les sources, ce fichier est plus facile à lire.

Le script **`configure`** s'est ici assuré qu'il disposait de **`gcc`**, **`install`**, et **`perl`**. Il a contrôlé certains fonctionnements du système de compilation et le contenu des fichiers d'en-têtes du répertoire /usr/include.

Quand il ne trouve pas ce qu'il cherche ou lorsqu'un test de bon fonctionnement échoue, **`configure`** peut désactiver une partie des fonctionnalités du logiciel, pour lesquelles la carence observée était indispensable. Il peut aussi contourner le problème. Par exemple, en l'absence de la commande **`install`** sur le système (ou s'il ne la trouve pas), **`configure`** la remplacera par le script install.sh.

Il lui arrive aussi d'abandonner devant l'adversité, signalant à l'utilisateur une erreur fatale qu'il devra régler lui-même.

Ces erreurs, très variées, ne sont hélas pas toujours très explicites. Par exemple, **`configure`** peut se plaindre de l'absence d'un symbole dans une bibliothèque ou d'un prototype de fonction d'un fichier d'en-têtes qui n'est pas conforme à ses vues. En clair, il est alors probable qu'une bibliothèque installée le soit sous une mauvaise version ; reste à mettre un nom et un numéro sur ces inconnues.

Ces situations sont rares : en général, tout se passe bien. Le problème le plus courant est celui de l'absence d'une bibliothèque (ou son installation en un endroit non standard). Des options du script **`configure`** permettent souvent de lui indiquer où trouver ce qu'il n'a pas pu découvrir seul.

S'il a été créé par **`autoconf`**, le script **`configure`** reconnaîtra l'option **`--help`** : il fournira alors la liste des options disponibles. Certaines sont génériques. Par exemple, l'option **`--prefix`** permet d'indiquer où le programme doit être installé (la valeur par défaut est /usr/local). D'autres sont spécifiques au programme. Exemple pour **`esound`** :

CULTURE **`esound`**

`esound` est un serveur capable de mixer plusieurs flux audio et de jouer le résultat.

```
$ cd esound-0.2.22
$ ./configure --help
Usage: configure [options] [host]
Options: [defaults in brackets after descriptions]
```

```
Configuration:
  --cache-file=FILE       cache test results in FILE
  --help                  print this message
  --no-create             do not create output files
  --quiet, --silent       do not print 'checking...' messages
  --version               print the version of autoconf that created configure
Directory and file names:
  --prefix=PREFIX         install architecture-independent files in PREFIX
                          [/usr/local]
  --exec-prefix=EPREFIX   install architecture-dependent files in EPREFIX
                          [same as prefix]
(...)
  --enable-alsa           use alsa if available [default=yes]
  --with-audiofile        include support for audiofile related utilities and functions
                              ARG = { yes | no | check } [default=check]
  --with-audiofile-prefix=PFX   Prefix where AUDIOFILE is installed (optional)
  --with-audiofile-exec-prefix=PFX Exec prefix where AUDIOFILE is installed (optional)
  --disable-audiofiletest       Do not try to compile and run a test AUDIOFILE program
  --with-libwrap          use tcp wrappers
(...)
```

Compilons, enfin !

Quand **`configure`** a fini son travail, il crée des fichiers Makefile dans les divers répertoires extraits des archives. Ce sont des fichiers de règles indiquant à la commande **`make`** comment compiler les sources.

`make` est un automate de compilation. Invoquée dans un répertoire, cette commande recherche un fichier Makefile pour savoir quoi faire. Ce dernier définit un certain nombre de variables (comme `CFLAGS`, qui indique les options à donner au compilateur) et un certain nombre de règles de dépendances, donnant les fichiers nécessaires à la compilation de chaque élément du logiciel. Cette liste de recettes permettra à **`make`** de compiler les sources. Exemple de Makefile simple :

```
CFLAGS= -Wall -Werror -ansi

hello:  hello.o
        cc $(CFLAGS) -o hello hello.o

hello.o:        hello.c hello.h
        cc $(CFLAGS) -c hello.c
```

Ce Makefile indique les options de compilation (variable `CFLAGS`), puis explique que pour créer hello, il faut avoir créé hello.o. La ligne qui suit indique comment construire hello à partir de hello.o. Une autre règle informe que hello.o dépend de hello.c et hello.h, et donne la commande permettant de créer hello.o. Attention : les retraits en début de ligne doivent être des tabulations.

Malheureusement, il existe plusieurs **`make`**, pas tout à fait compatibles entre eux. Certains logiciels supposent que la version de **`make`** disponible est GNU **`make`**, ce qui est vrai sur un système GNU/Linux, mais pas forcément ailleurs. Les Unix constructeur en proposent une autre version, et les BSD une troisième. Ceci ajoute donc parfois une dépendance, à savoir **`gmake`** (GNU **`make`**), qu'on trouvera à l'adresse : ftp://ftp.gnu.org/pub/gnu/make/. **`gmake`** a le bon goût de se compiler avec un autre **`make`**, mais il impose tout de même d'utiliser un tel programme, ce qui peut poser des problèmes d'œuf et de poule (appelés problèmes de *bootstrap* en informatique).

PLUS LOIN **Makefile et makefile**

En l'absence du fichier Makefile (avec majuscule), **`make`** cherchera un fichier makefile (sans majuscule – rappelons que sous Unix les majuscules ont leur importance dans les noms de fichiers).

PLUS LOIN **Génération des Makefile**

Le script **`configure`** génère les Makefile à partir de fichiers Makefile.in, modèles à trous. Certains projets adoptent également l'outil **`automake`**, qui permet de générer ces Makefile.in à partir de fichiers Makefile.am. Le but est d'éviter de recopier systématiquement de nombreuses règles imposées par les standards de distribution du projet GNU.

CULTURE **`make` et `gmake`**

Les BSD disposant de leur propre commande **`make`**, ils appellent **`gmake`** la version GNU de **`make`** quand il leur faut l'installer.

SCÉNARIO CATASTROPHE **En l'absence de `make`**

En l'absence de toute commande **`make`**, il faut la compiler à partir d'un script shell. C'est ainsi que démarre le processus de compilation croisée du système NetBSD.

Pour compiler, il suffit donc de taper **make** (ou **gmake**) après la création par **configure** de tous les Makefile. Pendant que **make** travaille, on obtient une sortie comme celle-ci (obtenue pendant la compilation de **pdksh**) :

```
CONFIG_FILES="" CONFIG_HEADERS=config.h ./config.status
creating config.h
config.h is unchanged
date > stamp-h
cc -c  -DHAVE_CONFIG_H -I. -I. -O2 alloc.c
cc -c  -DHAVE_CONFIG_H -I. -I. -O2 c_ksh.c
cc -c  -DHAVE_CONFIG_H -I. -I. -O2 c_sh.c
cc -c  -DHAVE_CONFIG_H -I. -I. -O2 c_test.c
cc -c  -DHAVE_CONFIG_H -I. -I. -O2 c_ulimit.c
cc -c  -DHAVE_CONFIG_H -I. -I. -O2 edit.c
./emacs-gen.sh ./emacs.c > tmpemacs.out
mv tmpemacs.out emacs.out
cc -c  -DHAVE_CONFIG_H -I. -I. -O2 emacs.c
cc -c  -DHAVE_CONFIG_H -I. -I. -O2 eval.c
cc -c  -DHAVE_CONFIG_H -I. -I. -O2 exec.c
cc -c  -DHAVE_CONFIG_H -I. -I. -O2 expr.c
cc -c  -DHAVE_CONFIG_H -I. -I. -O2 history.c
(...)
cc -c  -DHAVE_CONFIG_H -I. -I. -O2 version.c
cc -c  -DHAVE_CONFIG_H -I. -I. -O2 vi.c
cc  -o ksh alloc.o c_ksh.o c_sh.o c_test.o c_ulimit.o edit.o emacs.o  eval.o exe
c.o expr.o history.o io.o jobs.o lex.o mail.o  main.o misc.o missing.o path.o sh
f.o sigact.o syn.o table.o trap.o  tree.o tty.o var.o version.o vi.o
./mkman ksh ./ksh.Man > tmpksh.1
mv tmpksh.1 ksh.1
```

make compile chaque fichier source .c en fichier objet .o et termine par la liaison de ces derniers en l'exécutable **ksh**.

En principe tout doit bien se passer, mais des problèmes mineurs peuvent survenir à la compilation, qu'un débutant pourra parfois corriger seul. L'auteur a appris le C en corrigeant des erreurs de compilation dans des logiciels libres. Voici donc un petit guide des erreurs de compilation, qu'on consultera en cas de besoin car sa lecture linéaire s'avérera sans doute fastidieuse.

CULTURE **pdksh**

pdksh est un Korn Shell de domaine public. Pour travailler sur un Unix préhistorique avec un **ksh** semblable à celui de NetBSD et OpenBSD, c'est vers **pdksh** qu'il faut se tourner.

Fichiers .h introuvables, option **-I** à ajuster

PLUS LOIN **Fichiers d'en-têtes**

Les fichiers d'en-têtes se terminent en .h et se situent dans les répertoires /usr/include, /usr/local/include, /usr/X11R6/include, ou /usr/pkg/include (ce dernier chemin est spécifique à NetBSD, comme nous le verrons plus loin).
Ils indiquent respectivement les prototypes des fonctions des bibliothèques des répertoires /usr/lib, /usr/local/lib, /usr/X11R6/lib, ou /usr/pkg/lib. Le compilateur en a besoin pour savoir comment compiler les appels vers ces fonctions.

Un fichier d'en-têtes (.h) est introuvable. Il faut commencer par le localiser avec les commandes **locate**, ou à défaut **find**. S'il est présent sur le système mais n'a pas été trouvé lors de la compilation, c'est qu'une option de compilation **-I** manque : ces options indiquent au compilateur un répertoire où chercher de tels fichiers.

Le problème provient sans doute de l'oubli d'une option de **configure**. Si cette voie n'aboutit pas, essayez d'ajouter l'option **-I** adéquate dans le Makefile. À défaut, on peut toujours recopier la ligne de commande qui a provoqué l'erreur et lui ajouter le **-I** manquant.

Cas d'un programme qui échoue à l'inclusion d'un fichier tiff.h :

```
$ make
(...)
cc -O2 -Wall -Werror -c  image.c
image.c:34: tiff.h: No such file or directory
$ grep "tiff.h" image.c
#include <tiff.h>
```

Il faut trouver ce fichier et indiquer sa position au compilateur avec l'option **-I**, qu'on pourra ajouter au Makefile. On peut aussi retaper la commande de compilation à la main en y ajoutant l'option **-I** : la procédure de **make** ne recommence pas au début à chaque fois, mais exécute simplement les commandes qui n'ont pas encore réussi.

```
$ locate tiff.h
/usr/pkg/include/tiff.h
$ cc -O2 -Wall -Werror -I/usr/pkg/include -c image.c
$ make
cc -O2 -Wall -Werror -c  log.c
cc -O2 -Wall -Werror -c  main.c
(...)
```

Fichiers .h introuvables, il manque un autre programme

Lorsque le compilateur signale l'absence d'un fichier .h, le premier réflexe doit être de rechercher ce dernier avec **locate** ou **find**. Si le fichier manquant n'est pas présent sur le système, il peut s'agir d'un fichier d'en-têtes compris dans une dépendance que **configure** n'a pas contrôlée. Il faudra alors installer le logiciel manquant, pas toujours évident à identifier. La documentation peut être une bonne source d'idées pour résoudre un problème.

On peut aussi tenter de taper le nom du fichier manquant dans un moteur de recherche sur le Web, manœuvre qui avec un peu de chance mènera rapidement à la distribution du logiciel manquant.

Fichiers .h introuvables, option -D à ajuster

On peut aussi explorer une autre piste. Certaines fonctionnalités sont facultatives ou peuvent être réalisées de différente façons. Ce sont les directives de compilation, spécifiées au compilateur avec l'option **-D**, qui permettent de préciser le comportement à adopter. Si le compilateur se plaint de ne pas trouver un fichier

DANS LE DÉTAIL **Correction de Makefile**

Dans le Makefile, les options de compilation sont souvent passées via la variable `CFLAGS`. Même sachant cela, celui qui a opté pour l'intervention sur le Makefile peinera parfois à s'y retrouver dans les fichiers les plus complexes.

ATTENTION **Où suis-je ?**

Les programmes complexes sont souvent organisés avec plusieurs répertoires de sources hiérarchisés, que **make** visite dans le cadre de ses compilations. Lors de la saisie de la commande de compilation, assurez-vous de vous trouver dans le bon répertoire, abritant le fichier .c concerné.

EN PRATIQUE **Directives de compilation**

Les directives de compilation indiquent au compilateur quoi compiler lorsque des parties du code source sont facultatives.
L'option **-DINET_SOCKETS** lui enjoindra par exemple de prendre en compte le code compris entre les marqueurs de pré-traitement `#ifdef INET_SOCKETS` et les marqueurs `#endif` (ou `#else`) qui leur correspondent.

ASTUCE **Avec ou sans le copier/coller**

Lors de l'ajustement d'options de compilation sur des commandes de compilation, ne vous privez pas, si vous en disposez, de la possibilité de copier/coller toute une ligne : les lignes de commande impliquées seront bien plus longues que dans les exemples présentés ici ! Sinon, il est toujours possible de rediriger la sortie de **make** dans un fichier, qu'on transformera en script shell :

```
$ make
(...)
cc -O2   -Werror  -DHAVE_CONFIG_H -DSHAREPATH=\"/usr/share\"  -I/usr/pkg/include -I.
-c foobar.c
foobar.c:431: xutil.h: No such file or directory
$ make > /tmp/foo.sh
foobar.c:431: xutil.h: No such file or directory
$ cat /tmp/foo.sh
cc -O2   -Werror  -DHAVE_CONFIG_H -DSHAREPATH=\"/usr/share\"  -I/usr/pkg/include -I.
-c foobar.c
$ vi /tmp/foo.sh
(...)
$ sh /tmp/foo.sh
cc -O2   -Werror  -DHAVE_CONFIG_H -DSHAREPATH=\"/usr/share\"  -I/usr/pkg/include -I.
-I/usr/local/include  -c foobar.c
```

PLUS LOIN **Différence entre "foo.h" et ⟨foo.h⟩**

La directive **`#include "foo.h"`** indique d'inclure le fichier foo.h du répertoire courant. **`#include <foo.h>`** recherchera foo.h dans les répertoires d'en-têtes standard (souvent /usr/include), puis dans tous les répertoires indiqués au compilateur par les options **`-I`**).

CULTURE **foo bar baz qux....**

Les développeurs et administrateurs anglophones utilisent traditionnellement ces mots lorsqu'ils ont besoin de représenter quelque chose – ce sont des variables « méta-syntaxiques », utilisées à des fins d'exemples ou d'illustrations de syntaxes. Les Français leur préfèrent souvent des mots comme *toto*, *titi*... Le *Jargon File* (http://catb.org/~esr/jargon/html/) explique l'origine de ces mots, ainsi que de nombreux autres éléments de la culture des *hackers* – terme en rien agressif ou péjoratif à l'origine, comme expliqué dans ce lexique.

ALTERNATIVE **config.h**

Les options de compilation sont souvent rassemblées dans un fichier config.h, que vous penserez à consulter si vous ne les trouvez pas dans les Makefile.

RAPPEL **Si `grep` ne connaît pas `-r`**

La commande **`grep`** des Unix les plus anciens est assez fruste, mais on peut l'utiliser comme suit :

```
$ cd /usr/include
$ find . -type f -exec grep \
> tcsetattr {} \; -print
```

ncurses.h et que le fichier de sources où se produit l'erreur contient un extrait tel que celui qui suit, il s'agit peut-être d'un problème d'option **`-D`** :

```
#ifdef HAVE_NCURSES
   #ifdef HAVE_NCURSES_H
      #include <ncurses.h>
   #else
      #include <curses.h>
   #endif
   #include "include/ec_interface.h"
   #include "include/ec_interface_sniff_data.h"
   #include "include/ec_interface_plugins.h"

   extern WINDOW *data_source_win;
   extern WINDOW *plugin_window;

#endif
```

Si le fichier ⟨ncurses.h⟩ n'existe effectivement pas, cela signifie peut-être tout simplement que le système n'en a pas besoin. Il se peut aussi qu'il manque une dépendance facultative, dont le programme peut se passer.

Bref, rien de grave, mais **`configure`** s'est pris les pieds dans le tapis et a préparé un Makefile avec des options de compilation incorrectes.

Tentez de supprimer **`-DHAVE_NCURSES_H`** et/ou **`-DHAVE_NCURSES`** dans le fichier Makefile : les choses peuvent s'arranger. Inversement, si ⟨curses.h⟩ est introuvable, vous pouvez tenter d'y rajouter **`-DHAVE_NCURSES_H`** et **`-DHAVE_NCURSES`**.

Erreurs de syntaxe, fonctions ou structures non définies

Obtenir une erreur de syntaxe dans un programme réputé fonctionner peut être assez déconcertant. Le plus souvent, il s'agit plutôt d'une définition incorrecte. Probablement un problème de directive **`-D`** manquante ou indésirable, ou tout simplement une inclusion d'un fichier d'en-têtes introuvable : l'auteur du programme n'a pas besoin de ce fichier sur les systèmes où il a testé son programme, mais ces en-têtes sont indispensables sur le vôtre.

La commande **`grep`** sera d'un secours précieux dans ces situations : elle permet de retrouver rapidement la définition manquante, localisant ainsi le fichier d'en-têtes à inclure. Si par exemple la compilation se bloque sur l'utilisation de la fonction **`tcsetattr`**, on cherchera comme suit le fichier d'en-têtes qui la définit :

```
$ grep -r tcsetattr /usr/include/*
/usr/include/sys/termios.h: * Commands passed to tcsetattr() for setting the ter
mios structure.
/usr/include/sys/termios.h:int  tcsetattr __P((int, int, const struct termios *)
);
```

Le fichier manquant identifié, on peut l'ajouter à la main dans les sources (dans l'exemple : **`#include <sys/termios.h>`**). Si cette inclusion était conditionnée par un **`#ifdef`**, on pourra ajuster l'option **`-D`** concernée. On constate parfois dans les ajouts manuels que l'ordre des directives **`#include`** a une importance ; il faut alors tâtonner.

Bibliothèques manquantes, option **-L** à ajuster

Lors de la phase de liaison, le compilateur **cc** invoque l'éditeur de liens **ld** pour fabriquer un exécutable. **ld** est parfois invoqué directement par **make**. On lui indique un certain nombre de fichiers objet .o à lier, ainsi que des bibliothèques, précisées par l'option **-l**. Exemple de phase de liaison typique :

```
$ cc -o hello -lxutil -L/usr/local/lib hello.o subr.o
```

Cela précise que l'exécutable hello doit être lié à partir des objets hello.o et subr.o, en utilisant la bibliothèque xutil. Le compilateur cherchera cette dernière dans les chemins standard (typiquement /usr/lib) et dans les chemins indiqués par les options **-L**. Il la cherchera sous le nom libxutil.a pour une liaison statique, et libxutil.so pour une liaison dynamique.

Lorsqu'une bibliothèque manque lors de la liaison, il faut commencer par la chercher avec **locate** ou **find**. Si elle existe, c'est une option **-L** à ajouter dans l'appel à l'éditeur de liens. Avec l'exemple précédent (remarquez la transformation du nom de la bibliothèque : **-ltoto** fait appel à la bibliothèque stockée dans un fichier appelé libtoto.so ou libtoto.a) :

```
$ cc -o hello -lxutil hello.o subr.o
/usr/bin/ld: cannot find -lxutil
collect2: ld returned 1 exit status
$ locate libxutil
/usr/local/lib/libxutil.a
$ cc -o hello -lxutil -L/usr/local/lib hello.o subr.o
$
```

DANS LE DÉTAIL **Si l'on corrige le Makefile**

Comme lors de la retouche des options **-I** ou **-D**, il est possible, pour ajuster une option **-L**, de recopier la commande invoquée par **make** et de l'exécuter en ligne de commande. On peut aussi modifier le Makefile, sachant que les options fournies à l'éditeur de liens sont en général précisées dans une variable LDFLAGS. Hélas, c'est parfois plus compliqué…

ATTENTION **Option -L et liaison dynamique**

Si c'est un binaire dynamique qui est compilé, indiquer l'emplacement de la bibliothèque avec **-L** va permettre la création du binaire. Mais lors de son exécution, l'éditeur de liens dynamiques devra retrouver la bibliothèque. Il la cherchera dans /usr/lib, puis dans les chemins indiqués par la variable d'environnement LD_LIBRARY_PATH.
Il n'est pas très pratique de devoir systématiquement définir LD_LIBRARY_PATH pour exécuter les binaires dynamiques liés avec des bibliothèques situées en des endroits étranges. Heureusement, l'option **-Wl,--rpath** permet lors de la liaison du binaire d'indiquer des chemins à essayer pour retrouver la bibliothèque à l'exécution :

```
$ cc -o hello -lxutil -L/usr/local/lib hello.o subr.o
$ ./hello
Shared object "libxutil.so.0" not found
$ export LD_LIBRARY_PATH=/usr/local/lib
$ ./hello
It works!
$ unset LD_LIBRARY_PATH
$ ./hello
Shared object "libxutil.so.0" not found
$ cc -o hello -lxutil -L/usr/local/lib -Wl,--rpath=/usr/local/lib hello.o subr.o
$ ./hello
It works!
```

Attention, la variable LD_LIBRARY_PATH n'est pas universelle sous Unix. Sous MacOS X, elle s'appelle par exemple DYLD_LIBRARY_PATH.

PLUS LOIN **ldconfig**

Sur certains systèmes Unix, l'éditeur de liens dynamiques maintient un *cache* des bibliothèques présentes sur le système – cela lui permet de faire la liaison plus rapidement.
La commande **ldconfig** permet de remettre à jour ce *cache* lors de l'ajout de bibliothèques. Elle utilise éventuellement un fichier de configuration lui indiquant dans quels répertoires il faut chercher des bibliothèques : sur GNU/Linux et FreeBSD, c'est le fichier /etc/ld.so.conf.
L'usage de binaires au format ELF permet de bénéficier d'une localisation automatisée des bibliothèques dynamiques, et devrait rendre **ldconfig** obsolète. **ldconfig** n'est par exemple plus inclus dans les distributions récentes de NetBSD car il n'y sert plus à rien.

PLUS LOIN **La commande nm**

nm permet de lister la table des symboles d'une bibliothèque ou d'un programme. On y trouve des fonctions, dénotées par des `T`, des variables globales, dénotées par des `D`, et un certain nombre d'autres objets (détaillés dans la page **info** de **nm**, la page **man** étant incomplète). Les symboles de type `U` sont des références vers une autre bibliothèque ; la bibliothèque les utilise mais ne les définit pas.

Bibliothèques manquantes, autres raisons

Autres motifs de bibliothèque manquante : il manque une dépendance, ou la bibliothèque n'est pas nécessaire dans votre cas particulier. Cette dernière situation sera traitée en tentant de supprimer l'option **-l** provoquant la liaison. Si cela multiplie les symboles manquants, c'est que peut-être cette bibliothèque était nécessaire, ou qu'il faut la remplacer par une autre.

Symboles manquants

Ce sont les pires problèmes de compilation à traiter : l'éditeur de liens trouve toutes les bibliothèques indiquées, mais certains symboles ne sont pas définis.

```
$ cc -L/usr/local/lib -lxutil -o foo builtin.o main.o log.o
builtin.o: In function `do_exp':
builtin.o(.text+0x1f8): undefined reference to `exp'
builtin.o: In function `double_to_int':
builtin.o(.text+0x788): undefined reference to `floor'
builtin.o(.text+0x790): undefined reference to `ceil'
```

La commande **nm** permet d'explorer le contenu d'une bibliothèque :

```
$ cd /usr/lib
$ nm libm.so.0
0002e8bc A _DYNAMIC
0002e8b0 A _GLOBAL_OFFSET_TABLE_
         w _Jv_RegisterClasses
000368bc G _SDA2_BASE_
00036948 G _SDA_BASE_
0002fa20 A __bss_start
         w __cxa_finalize
         w __deregister_frame_info
0002e948 g __dso_handle
0002fef8 A __end
         U __errno
00012370 T __ieee754_acos
00012058 T __ieee754_acosf
(...)
```

Pour chaque symbole manquant, il faudra trouver la bibliothèque qui le définit. En l'absence du symbole `XDPSIsDPSEvent`, on pourra procéder comme suit :

```
$ for i in /usr/lib/*.a ; do nm $i |grep XDPSIsDPSEvent|grep -v ' U '>/dev/null \
> && echo $i ; done
$ for i in /usr/X11R6/lib/*.a ; do nm $i |grep XDPSIsDPSEvent|grep -v ' U '>/dev/null \
> && echo $i ; done
libdps.a
$ for i in /usr/X11R6/lib/*.so ; do nm $i |grep XDPSIsDPSEvent|grep -v ' U '>/dev/null\
> && echo $i ; done
libdps.so
```

Il ne reste plus qu'à ajouter les options **-l** et **-L** pour que la bonne bibliothèque soit utilisée dans l'édition des liens. Si cela ne suffit pas, c'est un problème de version

PLUS LOIN **Vous vous souvenez de sed ?**

Les amateurs de sensations fortes apprécieront ceci :

```
$ nm /usr/lib/*.a |sed -n '/\.a:$/{s/:$//; h;}; /[^U] XDPSIsDPSEvent/{g; p;}'
$ nm /usr/lib/X11R6/*.a |sed -n '/\.a:$/{s/:$//; h;}; /[^U] XDPSIsDPSEvent/{g; p;}'
libdps.a
$ nm /usr/lib/X11R6/*.so|sed -n '/\.so:$/{s/:$//; h;}; /[^U] XDPSIsDPSEvent/{g; p;}'
libdps.so
```

de la bibliothèque ; la documentation précisera les éventuelles dépendances sur les versions.

Si vraiment cela ne va pas...

La lecture de tous ces cas de figure peut inquiéter si elle n'est pas accompagnée de travaux pratiques. En pratique, les problèmes sont rares, et ce guide pourra rester au placard dans la plupart des cas.

Tout problème bloquant recevra probablement une solution dans les listes de diffusion : les projets de logiciels libres en réservent souvent une aux questions des débutants. La difficulté principale est de bien poser sa question sur la liste adéquate.

Les petits projets sont dépourvus de liste de diffusion et d'aide faute de masse critique d'utilisateurs. Dans ce cas, un courrier électronique à l'auteur s'avère souvent efficace.

Enfin, si pour compiler un logiciel libre sur votre système vous avez dû intervenir sur les fichiers distribués, il est bon de le signaler à l'équipe de développement. Cela évitera peut-être à un autre de subir les mêmes déboires.

PIÈGE À C... **Compilation et horloge**

Une horloge déréglée peut provoquer des erreurs inexplicables à la compilation, En effet, **tar** extrait les fichiers en conservant leurs dates de dernière modification. Si l'heure système est antérieure à certaines dates, les fichiers générés seront plus vieux que les fichiers de code source – ce que certains processus de compilation supportent très mal. Tout changement d'heure au cours d'une compilation exposera lui aussi à ce type de risques.

Installation, désinstallation

Une fois la compilation achevée, on peut installer. Il suffit souvent pour cela de taper **`make install`**. À moins d'avoir indiqué à **`configure`**, par le biais de l'option **`--prefix`**, un répertoire où vous avez le droit d'écrire (si par exemple il vous appartient), il faudra passer root pour taper **`make install`**. Une fois le programme installé, il pourra être exécuté, sous éventuelles réserves de réglage des variables `PATH`, et `LD_LIBRARY_PATH`.

Pour la désinstallation, les choses sont parfois plus complexes. Certains programmes se désinstallent en tapant **`make deinstall`** ou **`make uninstall`** depuis le répertoire des sources (qu'il faut donc avoir conservé tout ce temps), d'autres pas. Mais un programme compilé avec un préfixe standard (/usr/local ou /usr) noiera ses différents fichiers au milieu de ceux des autres logiciels, et il sera difficile de les identifier pour les effacer sans dommages collatéraux.

Une solution consiste alors à faire une installation de test pour obtenir la liste des fichiers :

```
$ mkdir /tmp/bidon
$ configure --prefix=/tmp/bidon
(...)
$ make
(...)
$ make install
(...)
$ su
Password:
# cd /usr/local
# rm `cd /tmp/bidon; find . -type f -print`
```

RAPPEL `PATH` **et** `LD_LIBRARY_PATH`

Les variables d'environnement `PATH` et `LD_LIBRARY_PATH` sont respectivement utilisées pour indiquer au shell où chercher les programmes à exécuter et pour indiquer à l'éditeur de liens dynamiques où chercher les bibliothèques partagées. Elles comprennent une liste de répertoires séparés par des **:** (deux points).

Cette méthode est évidemment à manier avec précaution, et ne fonctionnera qu'en choisissant les mêmes options de **`configure`** que le jour de l'installation (excepté bien sûr **`--prefix`**).

Dans le cas général, il est délicat de se débarrasser d'un programme installé dans /usr ou /usr/local. Variante plus sûre de l'installation de test : placer chaque programme dans un sous-répertoire de /usr/local, comme /usr/local/inn pour INN et /usr/local/samba pour Samba. Pour supprimer un programme, il suffira ainsi de détruire le sous-répertoire qui héberge son installation. On n'a ainsi aucun conflit de noms de fichiers entre différents programmes, et on peut installer plusieurs versions en parallèle, sous réserve de toujours utiliser des noms de sous-répertoires distincts (et explicites), comme /usr/local/inn-2.3.3 et /usr/local/inn-2.3.0.

CULTURE **INN, Samba**

INN (*Internet Network News*, ou nouvelles du réseau Internet) est un serveur de *news* pour Usenet. Samba est un serveur de fichiers et d'impression implémentant les protocoles Microsoft utilisés par Windows.

Cette dernière méthode permet de passer d'une version à l'autre d'un logiciel en minimisant l'interruption de service : pendant que l'on continue à utiliser la version précédente, on peut tranquillement configurer la version suivante. L'inconvénient majeur de cette approche est qu'il faut prendre garde à ne pas se prendre les pieds dans les variables `PATH` et `LD_LIBRARY_PATH` : le mélange des bibliothèques d'une version et des binaires d'une autre peut se révéler explosif.

Unix ne proposant pas en standard de base de données où chaque programme installé indiquerait la liste de ses fichiers, aucune solution n'est vraiment satisfaisante. Les systèmes de paquetages traitent notamment ces questions ; il est temps de les évoquer.

Les systèmes de paquetages

En explorant les arcanes de l'installation à la main, nous avons rencontré un certain nombre de problèmes pénibles à traiter :

- Où est le logiciel ? Trouver les codes sources de projets éparpillés sur le Web est assez fastidieux, surtout lorsqu'ils ont le mauvais goût d'être fortement interdépendants, et pas du tout centralisés dans leur distribution.
- La gestion des dépendances peut aller de la simple routine (la documentation indique qu'il faut **`perl5`** et **`libpng`** ; je les ai ; on passe à la suite), au pire cauchemar lorsqu'il faut gérer des monstres comme les bureaux GNOME ou KDE, qui dépendent de dizaines de logiciels différents.
- La recompilation peut nécessiter des ajustement aux sources pas toujours à la portée du premier venu, surtout sur des logiciels complexes.
- Il n'y a pas de bonne méthode pour désinstaller proprement un programme. Lorsque l'on efface un logiciel, on peut aussi casser un autre logiciel qui en dépendait à notre insu.

Le travail c'est la santé, ne rien faire c'est la conserver : un système de paquetages résout tous ces problèmes – et en introduit potentiellement d'autres. Voyons cela.

Installer des binaires ou compiler des sources ?

La plupart des systèmes de paquetages distribuent des binaires déjà compilés, disponibles sur un serveur centralisé : on les trouve facilement. Les dépendances seront installées automatiquement ou au pire, indiquées de manière exhaustive (tous les programmes requis étant évidemment disponibles via le système de paquetages). Les programmes déjà compilés affranchissent des problèmes de compilation. Enfin, le système de paquetages fournit une base de données indiquant quels fichiers ont été installés, ce qui permet une désinstallation facile.

Ce sont là les fonctionnalités minimales et indispensables. Examinons maintenant les inconvénients du système de paquetages sur la compilation à la main.

Sans phase de compilation, on ne peut pas indiquer des options particulières, non prévues dans le paquetage. Impossible par exemple de choisir où le programme doit s'installer. Plus problématique, un binaire peut fonctionner correctement avec une version d'une bibliothèque et pas du tout avec une autre, et pour de nombreuses raisons : si par exemple la taille d'un argument dans une fonction a changé, le programme pointera sur une mauvaise adresse et plantera. Le système de paquetages devra donc indiquer très rigoureusement les numéros de versions acceptables pour les dépendances.

Enfin, plus grave : le paquetage binaire prive de la possibilité de modifier les programmes, un des grands atouts des logiciels libres. Modifier (ou faire modifier) le code source permet de satisfaire une nécessité particulière, comme la consignation d'un événement particulier dans les journaux, éliminer un avertissement ennuyeux... les possibilités sont infinies, et pas toujours réservées aux experts. Le débutant pourra opérer des modifications simples et utiles, et progressera ainsi efficacement dans le langage de programmation utilisé (souvent, le langage C).

Les systèmes de paquetages des BSD tentent de trouver le juste milieu entre paquetage binaire et compilation, pour profiter du meilleur des deux mondes.

Les systèmes de paquetages des BSD

Les trois systèmes BSD ont des systèmes de paquetages assez semblables. FreeBSD fut le premier à mettre le sien au point ; comme il était bon, les deux autres s'en sont ensuite très fortement inspirés pour concevoir le leur.

La philosophie est la suivante : recompiler les sources, en automatisant le plus d'étapes possibles. On bénéficie ainsi des avantages de la compilation, en en éliminant les soucis. Le seul inconvénient par rapport au paquetage binaire est le temps de mise à disposition : compiler est plus lent qu'installer un binaire. Mais on conserve la souplesse dans le choix des options, la certitude d'avoir bien compilé pour les bibliothèques qu'on utilise, et la possibilité de modifier les sources si on le souhaite.

Le système de paquetages connaît les dépendances de chaque logiciel, et peut les installer automatiquement ; il peut télécharger les sources car il connaît leur emplacement. Informé des sommes de contrôle des sources, il peut contrôler

SUR LES AUTRES UNIX **Systèmes de paquetages**

Chaque Unix a son système de paquetages. GNU/Linux en compte même plusieurs, selon la distribution. Les plus courants sont les .sol sur Solaris ; *Red Hat Package Manager* (.rpm) dans de nombreuses distributions Linux : Red Hat, Mandrake, SuSE et quelques autres ; et *Debian* (.deb) pour d'autres, dont Debian GNU/Linux.

leur intégrité. Il sait quels correctifs (*patches*) appliquer aux sources pour que la compilation se déroule correctement. Enfin, il sait compiler les sources.

Tout peut donc se faire automatiquement, du téléchargement des sources à l'installation du programme. Il est possible d'arrêter le processus à n'importe quelle étape, pour par exemple modifier les sources avant de les compiler et installer. Il est aussi possible de produire des paquetages binaires que l'on peut réinstaller ailleurs, en retrouvant alors les problèmes afférents aux paquetages binaires.

Avant d'étudier tout cela, précisons un peu la terminologie, car elle a le mauvais goût de changer d'un système à l'autre.

Un peu de terminologie

Chez FreeBSD et OpenBSD, on parle de « ports » pour designer les programmes proposés par le système de paquetages. On parle même du « système de ports », et non pas du « système de paquetages ». Ce terme s'explique par le fait qu'il s'agit de logiciels « portés » sous FreeBSD ou OpenBSD.

NetBSD appelle « port » un portage de NetBSD sur une architecture donnée. Ce que FreeBSD et OpenBSD appellent « port », NetBSD l'appelle « paquetage ».

Plus compliqué : chez FreeBSD et OpenBSD, le terme « paquetage » désigne des programmes issus du système de ports, compilés et distribués sous forme binaire. NetBSD parle de « paquetages binaires ».

Dans le présent chapitre, c'est la terminologie NetBSD qui est adoptée (n'oubliez pas que l'auteur travaille pour NetBSD, cela crée des habitudes).

Exemples d'utilisation

Le système de paquetages est distribué sous la forme d'un fichier .tar.gz, appelé ports.tar.gz sous FreeBSD et OpenBSD, et pkgsrc.tar.gz sous NetBSD. Gros de plusieurs méga-octets, il contient le nécessaire à la compilation automatique de tous les paquetages. En janvier 2004, ces fichiers avaient les tailles suivantes :

Système	Taille	Nombre de paquetages
FreeBSD	22,3 Mo	10300
NetBSD	19,6 Mo	4500
OpenBSD	6,3 Mo	1900

Autre divergence : par défaut, le pkgsrc de NetBSD installe ses paquetages dans /usr/pkg, alors que les ports de FreeBSD et OpenBSD s'installent dans /usr/local. Ces réglages par défaut sont bien entendu configurables ; c'est tout l'intérêt de paquetages qui se recompilent.

Pour le reste, les trois systèmes sont très semblables, et la plupart des explications données sont valables pour tous. Il fallait en choisir un pour les exemples ; ce sera le pkgsrc de NetBSD.

La mise en service du système de paquetages est simple : il suffit d'avoir installé les outils de compilation et d'extraire le contenu de pkgsrc.tar.gz (ou ports.tar.gz)

ALTERNATIVE **Paquetages et utilisation hors-ligne**

Le système de paquetages des BSD présente un autre inconvénient : sa façon de télécharger les sources des logiciels via le réseau sera peu satisfaisante pour les machines isolées ou qui disposent seulement d'un lien RTC (téléphonique) pour se connecter à Internet.
Pour ces utilisations, le système de paquetages peut générer un script de téléchargement, qui, une fois exécuté sur une machine connectée à Internet, rapatriera toutes les archives nécessaires à la compilation. Il ne reste plus qu'à transférer ces fichiers sur la machine déconnectée, par exemple grâce à un CD-ROM ou une clef USB, et la compilation du paquetage et de ses dépendances pourra se faire.
L'autre façon de régler ce problème, c'est l'utilisation de CD-ROM contenant des paquetages binaires. Chaque BSD y va de sa solution : FreeBSD et OpenBSD proposent les paquetages essentiels sous forme binaire sur les CD-ROM d'installation (surtout pour i386, moins pour les autres plateformes). NetBSD propose une demi-douzaine de CD-ROM de paquetages incluant tous les paquetages binaires, mais uniquement pour i386. Pour tout ce qui n'est pas prévu officiellement, on peut toujours télécharger des paquetages binaires disponibles en FTP et graver son propre CD-ROM.

CULTURE **pkgsrc multi-systèmes**

Le fichier de NetBSD utilise plus d'octets par paquetage géré : cela s'explique par sa capacité partielle à fonctionner sur d'autres systèmes (Darwin, FreeBSD, OpenBSD, SunOS, GNU/-Linux, et IRIX). Tous les paquetages ne fonctionnent pas partout, mais le concept d'un système de paquetages indépendant du système comme du processeur est assez intéressant.

EN PRATIQUE **Où installer le système de paquetages ?**

Rien n'oblige à installer l'arborescence du système de paquetages à un endroit particulier. On la place en général sous /usr, mais un utilisateur, dépourvu des droits de root, pourra très bien installer des paquetages dans son répertoire personnel.

quelque part. On installe ainsi une arborescence où les paquetages sont classés par catégorie :

```
$ tar -xzf pkgsrc.tar.gz
$ ls -F pkgsrc
CVS/                converters/         japanese/           plan9/
Makefile            corba/              lang/               print/
Packages.txt        cross/              licenses/           security/
README              databases/          mail/               shells/
README-IPv6.html    devel/              math/               sysutils/
README-all.html     doc/                mbone/              templates/
README.html         editors/            meta-pkgs/          textproc/
archivers/          emulators/          misc/               time/
audio/              finance/            mk/                 wm/
benchmarks/         fonts/              net/                www/
biology/            games/              news/               x11/
cad/                graphics/           parallel/
chat/               ham/                pkglocate*
comms/              inputmethod/        pkgtools/
$ ls -F pkgsrc/shells
CVS/                bash2-doc/          pdksh/              static-tcsh/
Makefile            es/                 rc/                 tcsh/
README.html         esh/                scsh/               zsh/
ast-ksh/            mudsh/              standalone-tcsh/    zsh3/
bash2/              osh/                static-bash2/
$
```

Chaque répertoire de paquetage contient plusieurs fichiers. Tout l'automatisme repose sur des fichiers Makefile ; on installe un paquetage en tapant **`make install`**.

```
$ cd pkgsrc/shells/bash2
$ ls -F
CVS/         Makefile     README.html  patches/
DESCR        PLIST        distinfo
$ make install
=> bash-2.05b.tar.gz doesn't seem to exist on this system.
=> Attempting to fetch bash-2.05b.tar.gz from ftp://ftp.gnu.org/pub/gnu/bash/.
=> [1956216 bytes]
Connected to ftp.gnu.org.
220 GNU FTP server ready.

(...)
```

Le fichier Makefile du paquetage contient très peu d'informations : l'URL où télécharger les sources, les options à donner à **`configure`**, les paquetages dont ce paquetage dépend, et leurs versions minimales. En voici un exemple assez simple :

```
# $NetBSD: Makefile,v 1.11 2002/10/25 17:33:59 wiz Exp $
#

DISTNAME=       bvi-1.3.0.src
PKGNAME=        bvi-1.3.0
WRKSRC=         ${WRKDIR}/${PKGNAME}
CATEGORIES=     editors
MASTER_SITES=   ${MASTER_SITE_SOURCEFORGE:=bvi/}

MAINTAINER=     sakamoto@netbsd.org
HOMEPAGE=       http://bvi.sourceforge.net/
COMMENT=        Vi-like editor for binary files

USE_BUILDLINK2= YES
GNU_CONFIGURE=  YES
MAKE_ENV+=      SHELL=${SH}

.include "../../devel/ncurses/buildlink2.mk"
.include "../../mk/bsd.pkg.mk"
```

EN PRATIQUE **Travailler en root ou pas ?**

La phase d'installation requerra évidemment les droits de root. Si l'on compile sous un compte non privilégié, le système de paquetages demandera de taper le mot de passe de root au moment de l'installation.

Si ces saisies répétées vous fatiguent, il est possible de tout compiler en tant que root, mais ce n'est pas une très bonne habitude du point de vue de la sécurité : une erreur idiote dans un script exécuté pendant la compilation, et c'est l'incident d'exploitation. Imaginez par exemple qu'un script exécute la commande **`rm -Rf ${top}/etc`** et que, suite à un malheureux concours de circonstances, la variable **`${top}`** ne soit pas définie…

L'auteur n'a jamais rencontré un tel problème, mais préfère prévenir que guérir, et compile le moins possible en tant que root.

CULTURE **`bvi`**

`bvi` est un éditeur de binaires qui reprend les principes de **`vi`**.

CULTURE **Infrastructure pour le système de paquetages**

Les systèmes de paquetages des trois BSD ont une racine commune, dont ils divergent ensuite. À l'origine, ils s'appuyaient ainsi tous trois sur la commande **`make`** et sur le *Bourne shell*. Avec sa version 3.5, OpenBSD utilise désormais Perl, ce qui simplifie substantiellement les scripts mis en jeu, mais impose la présence de Perl dans le système de base – ce qui n'est pas le cas dans NetBSD et dans les versions de FreeBSD ultérieures à la 5.0.

Toute la complexité du système est concentrée dans le fichier bsd.pkg.mk, long fragment de Makefile dont la lecture est déconseillée aux âmes trop sensibles. On peut en effet y trouver ce type d'atrocités :

```
.if defined(USE_IMAKE) && ${_PREFORMATTED_MAN_DIR} == "man"
_IMAKE_MAN_CMD= ${AWK} '/^([^\/]*\/)*man\/([^\/]*\/)?cat[1-9ln]\/.*0(\.gz)?$$/ { \
        sect = $$0; n = match(sect, "/cat[1-9ln]");                    \
        sect = sprintf(".%s", substr(sect, n + 4, 1));                 \
        s = $$0; sub("/cat", "/man", s); sub("\.0(\.gz)?$$", sect, s);  \
        if (match($$0, "\.gz$$") > 0) { ext = ".gz";} else { ext = "";} \
        $$0 = sprintf("%s%s", s, ext);                                 \
        } { print $$0; }' |
.  else
_IMAKE_MAN_CMD=
.endif # USE_IMAKE>
```

Les développeurs qui maintiennent pkgsrc ont beaucoup de mérite, car **make** est loin d'être l'outil idéal pour mettre en œuvre l'infrastructure d'un système de paquetages. Il permet toutefois d'éviter d'introduire de nouveaux outils ou de nouveaux formats : c'est un outil bien connu, qui évite d'imposer l'apprentissage d'un nouveau jeu de commandes.

Dans le répertoire du paquetage **bvi**, on trouve encore :

- un fichier DESCR, avec une description longue du paquetage ;
- un fichier PLIST, contenant la liste des fichiers installés par le paquetage (certains paquetages le génèrent dynamiquement) ;
- un fichier distinfo, renfermant les sommes de contrôle ;
- un répertoire patches, avec les correctifs (*patches* en anglais) nécessaires pour compiler correctement sous NetBSD ;
- un fichier README.html, généré automatiquement et présentant le paquetage d'une façon conviviale au format des navigateurs Web.

Le pkgsrc de NetBSD propose plusieurs cibles pour la commande **make**, selon l'endroit où l'on veut s'arrêter :

make fetch	Téléchargement des archives contenant les sources.
make extract	Extraction des archives contenant les sources dans le répertoire work, créé dans le répertoire du paquetage.
make patch	Application des correctifs spécifiques pour compiler sous NetBSD.
make build	La compilation en elle-même, précédée de l'exécution de **configure** le cas échéant.
make install	Installation du logiciel.
make clean	Nettoyage des sources du répertoire work.
make deinstall	Désinstallation du paquetage.

Bien entendu, la règle **make install** appliquera dans l'ordre les étapes **fetch**, **extract**, **patch**, et **build** avant de procéder à l'installation elle-même...

À VOIR **Configuration du pkgsrc**

pkgsrc/mk/bsd.pkg.defaults.mk est un fichier donnant les réglages par défaut de différentes options pour **pkgsrc**. On peut placer des valeurs personnalisées pour ces options dans le fichier /etc/mk.conf : par exemple, placer les répertoires de compilation dans /tmp permettra de ne pas encombrer une autre partition.
C'est aussi par le biais de ce fichier que l'on peut préciser les miroirs à utiliser pour les téléchargements sur des sites tels que SourceForge, le projet GNU, ou SunSite.

EN PRATIQUE **Comment utiliser ce qu'on vient d'installer**

C'est un problème courant : après l'installation d'un paquetage, on ne sait pas comment utiliser le nouveau programme.
L'examen du fichier PLIST peut s'avérer utile : on y trouve la liste des fichiers installés (les chemins sont relatifs à /usr/X11R6 ou /usr/pkg). On peut ainsi trouver les noms des programmes à invoquer et des pages **man** à consulter.

Bon d'accord, mais demain ?

Passons à la gestion quotidienne du pkgsrc. L'utilisateur peut éventuellement s'interroger sur la nature et l'emplacement des paquetages installés. Cette information est stockée dans /var/db/pkg : chaque paquetage y dispose d'un répertoire portant son nom et son numéro de version, contenant quelques fichiers donnant la liste des fichiers installés par ce paquetage, des paquetages qui en dépendent, et des paquetages dont il dépend.

Pour un accès rapide, les informations les plus utiles sont également présentes dans une base binaire : /var/db/pkg/pkgdb.byfile.db. Construite à partir des informations des fichiers texte, elle peut être entièrement régénérée si nécessaire.

Mise à jour des paquetages

Des trous de sécurité sont régulièrement découverts dans les paquetages, ce qui devrait fortement motiver l'administrateur à les mettre à jour.

Pour mettre à jour les paquetages, il faut mettre à jour le pkgsrc. La méthode la plus simple et la moins efficace est de télécharger un nouveau pkgsrc.tar.gz, par lequel remplacer l'ancien. On peut aussi procéder via CVS ou SUP ; la documentation sur le site Web fournira les détails. En quelques mots, pour CVS :

```
$ cd /usr/pkgsrc
$ export CVSROOT=:pserver:anoncvs@anoncvs.netbsd.org:/cvsroot
$ cvs login
Logging in to :pserver:anoncvs@anoncvs.netbsd.org:2401/cvsroot
CVS password: anoncvs
$ cvs update -P -d
```

Le pkgsrc mis à jour, il existe deux manières de mettre les paquetages à jour. La méthode propre consiste à taper **`make update`** dans un répertoire de paquetage : cette commande a pour effet de mettre à jour le paquetage et tous ceux qui en dépendent. On commence par une désinstallation massive, on recompile tout, et on réinstalle. En théorie, c'est parfait, mais le moindre incident en cours de route laissera en plan un pkgsrc ravagé, et très incomplet. On maniera donc cette méthode avec précaution, surtout pour les paquetages dont dépendent beaucoup d'autres.

Une méthode plus douce, **`make replace`**, a pour effet d'archiver l'ancien paquetage et d'installer le nouveau sans toucher aux dépendances. La compilation n'étant pas complète, certaines dépendances pourront ne plus fonctionner avec la nouvelle version, mais au moins il sera facile de revenir en arrière.

DORMEZ TRANQUILLE **`audit-packages`**

Pour aider à découvrir les paquetages qui ont besoin d'une mise à jour de sécurité, NetBSD propose le paquetage security/audit-packages. Une fois installé, il télécharge chaque jour une liste de vulnérabilités connues et indique quels paquetages installés sont vulnérables. L'administrateur reçoit dans sa boîte aux lettres l'alerte de sécurité avec le rapport quotidien du système.

OpenBSD et FreeBSD traitent ce problème différemment. Pour OpenBSD, la page http://www.openbsd.org/pkg-stable.html recense les mises à jour de sécurité des paquetages. Chez FreeBSD, le système de paquetages est traité comme le système de base, et des alertes de sécurité sont émises lors de la découverte de vulnérabilités dans les paquetages.

PIÈGE À C... **Mais ça n'avance pas !**

La commande **`pkg_add`** télécharge en silence, ce qui peut donner l'impression qu'elle reste bloquée. L'option **`-v`** lui enjoindra d'être plus loquace.

PERFORMANCES **OpenOffice.org**

pkgsrc propose deux versions d'OpenOffice.org : la version native, compilée pour NetBSD, et la version Linux, qui emploie la couche de compatibilité binaire Linux de NetBSD.

Au moment où sont écrites ces lignes, la version Linux fonctionne plus rapidement que la version native, faute d'une optimisation correcte.

Les paquetages binaires

L'administrateur lassé de compiler ou pressé (la compilation d'OpenOffice.org prend plus de 24 heures sur une machine récente) passera par un paquetage binaire. On les trouve sur le site FTP du projet, dans le répertoire /pub/NetBSD/packages. Au format .tar.gz, ils ne contiennent que les fichiers binaires. On les utilise avec les commandes **`pkg_add`** (auquel on peut fournir directement l'URL d'un paquetage binaire sur le serveur FTP) et **`pkg_delete`**, pour respectivement les ajouter ou les ôter. Exemple d'utilisation :

```
# pkg_add ftp://ftp.netbsd.org/pub/NetBSD/packages/1.6/i386/wm/wmakerconf-2.8.1nb1.tgz
```

Les paquetages pré-compilés sont des archives au format .tar contenant les binaires et quelques fichiers de méta-informations dont les noms commencent par +. On peut les installer avec la commande **tar**, mais une telle installation ne sera pas enregistrée dans la base des paquetages installés.

```
$ tar -tvzf wmakerconf-2.8.1nb1.tgz
-rw-r--r--  1 root     wheel       5549 Oct 28 15:50 +CONTENTS
-rw-r--r--  1 root     wheel         47 Oct 28 15:50 +COMMENT
-rw-r--r--  1 root     wheel        701 Oct 28 15:50 +DESC
-rw-r--r--  1 root     wheel       9719 Oct 28 15:50 +MTREE_DIRS
-rw-r--r--  1 root     wheel        498 Oct 28 15:50 +BUILD_VERSION
-rw-r--r--  1 root     wheel       2080 Oct 28 15:50 +BUILD_INFO
-rw-r--r--  1 root     wheel          7 Oct 28 15:50 +SIZE_PKG
-rw-r--r--  1 root     wheel          9 Oct 28 15:50 +SIZE_ALL
-r-xr-xr-x  1 root     wheel       6120 Oct 28 15:50 bin/mkpreview
-r-xr-xr-x  1 root     wheel     251132 Oct 28 15:50 bin/wmakerconf

(...)
```

On peut recourir aux paquetages binaires pour compiler une fois et déployer sur un parc de machines identiques, ce qui fait gagner un temps précieux. pkgsrc peut créer de tels paquetages avec la cible **make package**.

ASTUCE **Manipuler les fichiers dont le nom commence par le caractère +**

Taper **more +COMMENT** ne provoquera qu'un message d'erreur. En effet, pour les commandes de shell, + et – précèdent souvent des options. **more** croit donc être invoquée avec l'option **+COMMENT**, inconnue, et sans arguments.

more '+COMMENT' aura le même effet : il ne s'agit pas d'un problème d'interprétation de caractères spéciaux par le shell, mais d'un problème de compréhension de son argument par la commande. La solution consiste à indiquer un nom de fichier ne commençant pas par un caractère introduisant classiquement une option : **more ./+COMMENT**.

Logiciels propriétaires

Malheureusement, tous les programmes ne sont pas libres. Netscape, la machine virtuelle Java, RealPlayer, Opera, les anti-virus, MatLab, Mathematica, Oracle... ne sont notamment pas distribués sous forme de code source.

Comment les faire fonctionner ?

Pour qui c'est compilé ?

Ces programmes propriétaires sons souvent distribués sous forme de binaires par les sociétés qui les produisent. Ces binaires, produits pour les plateformes les plus populaires, excluent presque toujours les systèmes BSD. GNU/Linux, bien plus répandu, a en revanche souvent droit à sa version binaire. On trouve aussi des logiciels propriétaires intéressants sur les Unix constructeur. Sur IRIX, par exemple, on peut trouver Photoshop ou AutoCAD, mais aussi d'importants programmes de modélisation et de calcul scientifique, moins connus.

En l'absence de binaires spécifiques, utilisons ceux des autres...

Compatibilité binaire

Pour cela, les systèmes BSD ont développé des couches de compatibilité binaire, qui permettent de faire fonctionner de façon transparente et sans perte de performance des programmes compilés pour le même processeur et pour des systèmes d'exploitation différents. Le principe est simple : le programme fonctionne normalement, et tout appel système est intercepté et interprété comme le ferait le

PLUS LOIN **Appels système**

Les appels système sont les points d'entrée du noyau, et sont utilisés par les processus pour avoir accès aux ressources qu'il protège. Ainsi, l'appel système **open()** permet d'ouvrir un fichier, **execve()** invoque un nouveau programme, **pipe()** crée un tuyau...

Lorsqu'un processus invoque un appel système, son exécution est suspendue, et le contrôle est passé au code présent dans le noyau pour traiter sa requête. Une fois l'appel système traité, le processus peut reprendre la main. La seule chose visible du point de vue du processus, c'est le comportement de l'appel système, pas son implémentation. On peut donc remplacer le noyau pour lequel le processus a été compilé par un autre – moyennant d'émuler correctement le comportement des appels système du noyau d'origine. C'est ainsi que fonctionne la compatibilité binaire.

Pour plus d'informations sur l'implémentation des couches de compatibilité binaire, vous pouvez consulter la série d'articles écrite par l'auteur pour ONLamp.com, en anglais : http://www.onlamp.com/pub/a/bsd/2002/08/08/irix.html

noyau du système cible de la compilation. Ainsi, le programme s'exécute innocemment, pensant se trouver sur le système original. Il fonctionne à pleine vitesse puisque son code n'est pas émulé.

FreeBSD peut ainsi utiliser les binaires des systèmes suivants :

Linux/i386	SCO/i386	SunOS/i386
Linux/alpha	OSF1/alpha	

OpenBSD, quant à lui, est un peu plus riche :

Linux/i386	FreeBSD/i386	NetBSD/i386
HPUX/m68k	BSD-OS/i386	SunOS/i386
OSF1/alpha	SunOS/sparc	

NetBSD étant le plus multi-plateformes des trois, il n'est pas étonnant que ce soit le plus complet en la matière : c'est celui qui s'est frotté au plus grand nombre de systèmes. Il propose ainsi :

Linux/i386	Linux/alpha	Linux/m68k
Linux/powerpc	Linux/mips	Linux/arm
BSD-OS/i386	FreeBSD/i386	SCO/i386
SunOS/i386	SunOS/sparc	IRIX/mips
OSF1/alpha	Ultrix/vax	Ultrix/mips
Darwin/powerpc	HPUX/m68k	

Certaines compatibilités binaires sont bien plus avancées que d'autres. Par exemple, la couche de compatibilité Linux de FreeBSD permet d'employer de façon fiable les binaires d'Oracle, Mathematica, ou Matlab. Celles de NetBSD et OpenBSD sont un peu moins matures : ces programmes n'ont pas encore été signalés comme fonctionnant totalement (par exemple, MatLab pour Linux/i386 fonctionne sous NetBSD/i386, moyennant de l'invoquer avec l'option **`-nojvm`**). D'autres programmes pour Linux/i386, tels que Netscape, RealPlayer, ou la machine virtuelle Java de Sun fonctionnent très bien sur NetBSD/i386.

Certaines couches de compatibilité binaire sont encore dans un état expérimental, soit faute d'utilisateur depuis longtemps (compatibilité Ultrix), soit parce qu'elles sont en cours de développement (compatibilité Darwin).

Ces choses évoluent rapidement, et il se peut que ces appréciations ne soient plus d'actualité lorsque vous lirez ces lignes. En cas de besoin d'une application particulière, le plus sûr est de poser la question précise sur les listes de diffusion adéquates.

PERFORMANCES **Compatibilité binaire ou émulation**

Avec la compatibilité binaire, le code de l'application est exécuté tel quel par le processeur. Les émulateurs permettent aussi de faire fonctionner les programmes prévus pour d'autres systèmes, mais en analysant le code de l'application et en le traduisant en code utilisable par le processeur.

L'émulation demandant une étape de traduction, elle est plus lente. Par contre, elle permet de faire fonctionner des programmes écrits pour un autre processeur. La compatibilité binaire se limite pour sa part aux programmes compilés pour des systèmes d'exploitation différents, mais pour le même processeur.

Interaction avec le système de paquetages

La plupart des systèmes d'exploitation fonctionnent avec des binaires dynamiques, qui ont besoin pour fonctionner de bibliothèques dynamiques. Les couches de compatibilité binaire n'émulant que le comportement du noyau, il est nécessaire d'installer les bibliothèques du système dont on souhaite utiliser des binaires.

Pour éviter de mélanger les bibliothèques natives avec les éléments des systèmes étrangers, les couches de compatibilité binaire réservent une arborescence dédiée aux bibliothèques et fichiers de ces derniers. Ainsi, sous NetBSD et OpenBSD, les binaires GNU/Linux chercheront toujours leurs fichiers relativement à `/emul/linux` avant de les chercher sous la racine. Sous FreeBSD, ces arborescences dédiées sont placées sous `/compat`.

Dans le cas des systèmes propriétaires comme SunOS ou IRIX, il faudra copier les bibliothèques à la main. Des pages **`man`** comme `compat_sunos(8)` peuvent expliquer la marche à suivre.

Pour les systèmes libres et les systèmes propriétaires dont les éditeurs laissent les bibliothèques en téléchargement, il suffit d'utiliser les paquetages prévus pour cela : ils s'acquitteront de ces tâches. Par exemple, dans le `pkgsrc` de NetBSD, la catégorie `emulators` contient un certain nombre de paquetages pour installer les bibliothèques pour GNU/Linux ou FreeBSD.

`pkgsrc` contient aussi des paquetages qui installent les binaires GNU/Linux de Netscape ou Opera. Ils dépendent des paquetages de `pkgsrc/emulators` installant les bibliothèques de GNU/Linux. Ainsi, taper **`make install`** dans `pkgsrc/www/communicator` sur NetBSD/i386 installera *Netscape Communicator* pour GNU/Linux/i386 ainsi que toutes les bibliothèques GNU/Linux nécessaires à son fonctionnement.

Récupérer les paquetages des autres

Il est fréquent, lors de l'utilisation de la compatibilité binaire, de tomber nez à nez avec un paquetage du système étranger. Comment installer un `.rpm` sur un système BSD, par exemple ?

> ATTENTION **Problèmes de licences**
>
> Les couches de compatibilité binaires permettent de faire fonctionner des logiciels propriétaires comme Photoshop ou FrameMaker sur les systèmes BSD, mais n'oubliez pas que ces logiciels ne sont pas libres : il faut disposer d'une licence pour les utiliser légalement.
> Lors de la copie des bibliothèques d'un système d'exploitation pour les utiliser avec la compatibilité binaire, le même problème se pose : si le système d'exploitation est propriétaire, il faut disposer d'une licence pour utiliser ses bibliothèques.
> Ces problèmes ne se posent pas dans le cas de logiciels ou bibliothèques libres, écrits notamment pour permettre et faciliter ce genre de transferts.

On peut toujours utiliser les outils du système étranger, et recourir au programme **rpm** de GNU/Linux pour installer les fichiers .rpm. Cela fonctionne, mais on importe aussi les inconvénients du système de paquetages étranger : il faut subir sa gestion des dépendances et sa base de paquetages installés.

L'alternative consiste à extraire les fichiers de l'archive .rpm avec des outils natifs – en perdant évidemment la gestion des dépendances et la désinstallation.

Installer un .rpm

L'affaire est assez complexe, les versions différentes abondent. Pour les versions 1 et 2, le programme **rpm2pkg** (catégorie pkgtools sur NetBSD) fera l'affaire. Pour les versions 3 et 4, il faudra recourir au script **rpm2cpio.pl**, (catégorie converters sur NetBSD et OpenBSD, et archivers sur FreeBSD).

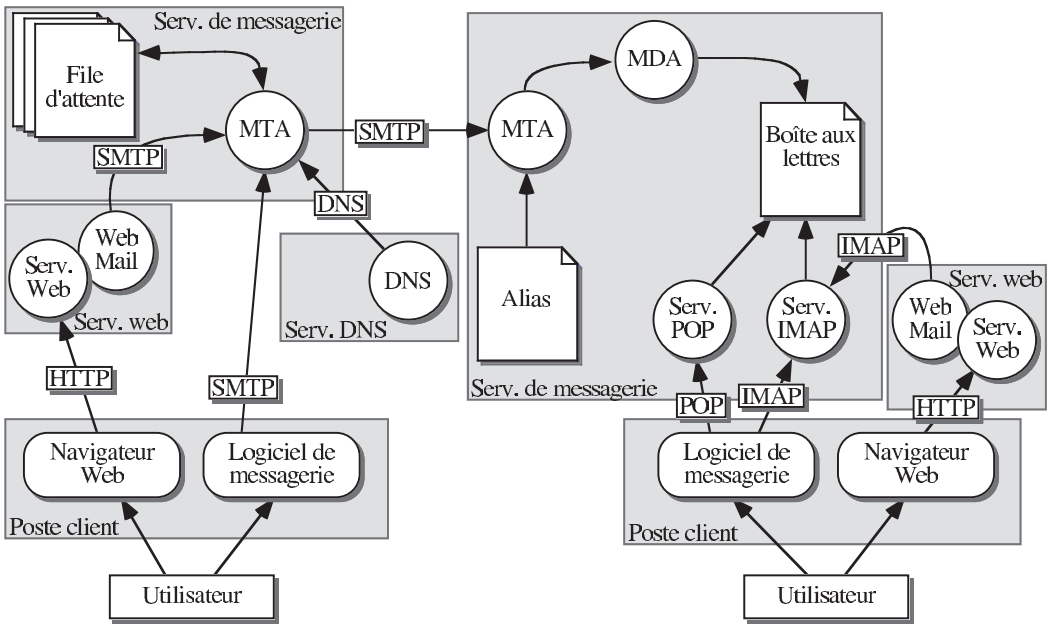
rpm2cpio.pl convertit le .rpm en archive **.cpio** ; on finira alors le travail avec la commande **pax** :

```
$ cat paquetage.rpm | rpm2cpio.pl | pax -r
```

CULTURE **Useless Use Of Cat**

L'emploi de **cat** dans les lignes de commande est souvent inutile, en particulier quand sa sortie est dirigée dans l'entrée standard d'une autre commande à l'aide d'un tube (|). Les spécialistes pointilleux évitent ces *Useless Uses of Cat* car elles créent un processus inutile. On peut les remplacer par une redirection de l'entrée standard de la commande qui suit, comme ceci : **rpm2cpio.pl < paquetage.rpm | pax -r**.
Elles sont toutefois utiles pour détailler les choses et les rendre plus lisibles aux débutants.

12


Serv. de messagerie
File d'attente
MTA
SMTP
SMTP
DNS
Web Mail
Serv. Web
Serv. web
DNS
Serv. DNS
HTTP
SMTP
Navigateur Web
Logiciel de messagerie
Poste client
Utilisateur
MTA
MDA
Boîte aux lettres
Alias
Serv. de messagerie
Serv. POP
Serv. IMAP
IMAP
POP
IMAP
Serv. web
Web Mail
Serv. Web
HTTP
Logiciel de messagerie
Navigateur Web
Poste client
Utilisateur

Tout pour le serveur : Web, DNS, et messagerie

Nous avons vu au chapitre 8 comment configurer des services simples comme DHCP ou FTP. Passons à plus complexe : le Web, le DNS, et la messagerie. Ce chapitre se focalisant plus sur les applications que sur le système, il s'appliquera assez facilement à n'importe quel Unix.

SOMMAIRE

MOTS-CLEFS

- Serveur DNS
- Serveur Web
- Messagerie

Serveur Web

Le serveur Web le plus répandu est de loin Apache, utilisé par plus de 60 % des serveurs Web sur Internet. Comme de plus il est assez simple à configurer, c'est celui que nous examinerons.

Installation d'Apache

Apache est inclus dans la distribution de base d'OpenBSD, et compris dans le système de paquetages de NetBSD et FreeBSD, catégorie www. On peut aussi l'installer à partir des sources, situées sur le site Web du projet Apache : http://www.apache.org.

Des modules

Apache est conçu de façon très modulaire. Son cœur se contente de servir des pages Web, et la plupart de ses fonctionnalités sont disponibles via des modules – qui peuvent ou non faire partie de la distribution d'Apache. Quelques exemples plus ou moins courants :

- **mod_ssl** fournit les fonctionnalités de chiffrement SSL, pour construire des sites Web sécurisés. Ces sites sont accessibles par les URL commençant en https://
- **mod_php** embarque l'interpréteur PHP dans Apache. Utilisé pour des pages à contenu dynamique, beaucoup d'applications Web y ont recours.
- **mod_perl** embarque l'interpréteur Perl dans Apache.
- **mod_xslt** rend Apache capable d'effectuer des transformations XSLT. Ce module permettra de publier des documents XML, qui seront transformés en HTML par des feuilles de style XSL.
- **mod_proxy** permet à Apache de se comporter en mandataire HTTP (*proxy* en anglais).

Certains modules peuvent être compilés sous forme statique ou dynamique. Les modules dynamiques prennent la forme de fichiers .so, que le serveur Apache pourra charger à la volée. D'autres doivent être compilés statiquement : c'est le cas de **mod_so**, qui charge les modules dynamiques.

En installant Apache depuis les sources, il faudra faire ce choix pour chaque module qui le propose. Si vous installez Apache par le système de paquetages, tout se fera tout seul : le concepteur du paquetage a déjà choisi ce qui est statique ou dynamique. Une fois Apache installé, on pourra consulter la liste des modules compilés statiquement :

```
$ httpd -l
Compiled-in modules:
  http_core.c
  mod_vhost_alias.c
  mod_env.c
  mod_log_config.c
  mod_mime_magic.c
  mod_mime.c
  mod_negotiation.c
```

CULTURE **Popularité des serveurs Web**

Le site http://www.netcraft.com produit régulièrement un rapport sur la popularité des différents serveurs Web (le terme « part de marché » est inapproprié dans la mesure où le produit le plus utilisé est gratuit…). Depuis début 1996, Apache est la solution la plus populaire, suivie de loin par IIS de Microsoft.

ALTERNATIVE **D'autres serveurs Web**

Dans la sphère du logiciel libre, les autres solutions sont peu nombreuses. NCSA **httpd**, l'ancêtre d'Apache, a pratiquement disparu désormais.
Pour exécuter des applications Web par-dessus Apache, il conviendra d'envisager Zope.

EN PRATIQUE **Modules Apache dans pkgsrc**

Dans le système de paquetages de NetBSD, les modules Apache sont préfixés par ap_ et non par mod_. Ainsi, **mod_ssl** s'installe par le paquetage pkgsrc/www/ap_ssl.

CULTURE **XSL**

XSL est un dialecte XML prévu pour les transformations de documents XML : transformation en XHTML (la version XML de HTML), en texte pur, etc. On peut par exemple utiliser XSL pour modifier dynamiquement un dessin vectoriel au format SVG.

PLUS LOIN **Pourquoi des modules dynamiques**

Les modules dynamiques sont appréciables pour utiliser des fonctionnalités non fournies directement par Apache. PHP est un bon exemple : on le met plus souvent à jour qu'Apache, pour corriger un trou de sécurité ou pour bénéficier d'une nouvelle fonctionnalité.
En compilant PHP comme module, on n'a pas besoin de recompiler Apache pour mettre à jour PHP : il suffit de recompiler le module PHP et de redémarrer Apache.

```
mod_status.c
mod_info.c
mod_include.c
mod_autoindex.c
mod_dir.c
mod_cgi.c
mod_asis.c
mod_imap.c
mod_actions.c
mod_speling.c
mod_userdir.c
mod_alias.c
mod_rewrite.c
mod_access.c
mod_auth.c
mod_auth_anon.c
mod_auth_db.c
mod_digest.c
mod_cern_meta.c
mod_expires.c
mod_headers.c
mod_usertrack.c
mod_unique_id.c
mod_so.c
mod_setenvif.c
```

Invocation d'Apache

Lors de l'administration du serveur Apache, il faut manipuler plusieurs fichiers et programmes, dont l'emplacement varie d'une installation à l'autre – mais vous en savez maintenant assez pour les retrouver seul :

- **`httpd`** est le serveur lui-même. On l'invoque rarement directement.
- **`apachectl`** permet de démarrer, stopper, ou redémarrer le serveur Apache, ainsi que d'en vérifier le fichier de configuration.
- httpd.conf, fichier de configuration du serveur Apache, centralise toute sa configuration.
- access_log est le journal des requêtes traitées.
- error_log est le journal des erreurs de traitement survenues.

On invoque le serveur avec **`apachectl start`**. On souhaitera probablement modifier le fichier httpd.conf pour changer la configuration par défaut.

C'est httpd.conf qui indique quels modules charger, où se trouvent les pages Web à servir, sous quelle forme enregistrer les informations dans les journaux, et bien d'autres choses encore.

Serveurs virtuels

Il est assez courant d'utiliser la même machine pour servir plusieurs adresses ou sites différents. On peut par exemple faire pointer les adresses `www.example.com` (serveur institutionnel) et `www.support.example.com` (serveur de département) sur la même machine, pour économiser du temps et de l'argent.

SÉCURITÉ **Quel UID pour Apache ?**

On évite en général de donner les droits de root aux *daemons* qui n'en ont pas besoin, afin de minimiser l'effet d'un éventuel trou de sécurité.

Apache ne fait pas exception à cette règle, et fonctionne sous l'UID d'un utilisateur aux droits restreints. Les noms varient : `nobody`, `apache`, `httpd`, ou `www`, mais le principe demeure.

Le fichier httpd.conf indique l'utilisateur sous lequel Apache doit fonctionner. Assurez-vous qu'il existe, et créez-le dans le cas contraire. Il faudra ensuite s'assurer que cet utilisateur a bien le droit de lire les fichiers correspondant aux pages Web.

ALTERNATIVE **Vieux Apache**

Sur les anciennes versions d'Apache, la commande **`httpdctl`** remplace **`apachectl`**.

Le fichier de configuration httpd.conf fut un temps flanqué de compagnons srm.conf et access.conf, mais il centralise tout désormais.

PIÈGE À C… **Le paramètre à configurer absolument**

Lorsqu'il répond à une requête, le serveur renvoie son nom, par défaut le nom de la machine (exemple : `rominet`). En général, on préfère que le serveur Web soit connu sous l'alias classique `www.example.net`, et surtout pas `rominet.example.net` (son nom courant sur le domaine suivi du nom du domaine).

Le paramètre le plus essentiel de la configuration est donc sans doute la directive **`ServerName`**, qui indique sous quel nom le serveur doit se présenter.

Bien entendu, le DNS sera configuré de telle façon que `www.example.net` renvoie sur `rominet.example.net`.

On indique une telle configuration dans le fichier httpd.conf grâce à la directive **VirtualHost** :

```
<VirtualHost 192.0.2.15>
ServerName "www.example.com"
DocumentRoot "/www/main"
</VirtualHost>
<VirtualHost 192.0.2.16>
ServerName "www.support.example.com"
DocumentRoot "/www/support"
</VirtualHost>
```

Bien entendu, le serveur doit alors porter les deux adresses IP sur une interface réseau, et les noms DNS `www.example.com` et `www.support.example.com` pointeront tous deux vers les bonnes adresses.

Cette méthode, simple à comprendre, ne conviendra pas aux institutions en manque d'adresses IP. C'est pourquoi on peut configurer des serveurs virtuels en se fondant sur l'adresse DNS : toutes les adresses DNS pointeront sur la même IP, et c'est le serveur qui décidera comment servir chaque requête :

```
<VirtualHost *>
ServerName "www.example.com"
DocumentRoot "/www/main"
</VirtualHost>
<VirtualHost *>
ServerName "www.support.example.com"
DocumentRoot "/www/support"
</VirtualHost>
```

ASTUCE **La documentation d'Apache**

La documentation d'Apache est en principe installée sur la machine ; vous pourrez donc la consulter localement, à l'adresse http://localhost/manual – à condition bien sûr que votre serveur Web fonctionne.

On peut encore configurer des serveurs distincts sur différents ports de la même adresse IP. La documentation d'Apache détaille tout cela à l'adresse http://httpd.apache.org/docs.

Serveurs Web sécurisés

Les serveurs Web sécurisés permettent de chiffrer le trafic HTTP dans des sessions SSL. Ils répondent par défaut sur le port 443 plutôt que sur le port 80, et ils se distinguent par des URL en https://. Cela ne peut évidemment fonctionner que de concert avec un client qui comprend le SSL, mais c'est le cas de tous les navigateurs modernes.

EN PRATIQUE **Documentation de `mod_ssl`**

La documentation des options SSL du httpd.conf est disponible sur le site Web de **mod_ssl**, à l'adresse suivante : http://www.modssl.org.

SSL nécessite un serveur Apache compilé avec **mod_ssl** (statiquement ou dynamiquement), un certificat SSL, et bien sûr la bonne configuration dans httpd.conf.

Votre installation d'Apache avec **mod_ssl** devrait inclure un fichier httpd.conf.default contenant toutes les options nécessaires au bon fonctionnement de SSL : utilisez-le comme modèle.

Une fois le fichier httpd.conf correctement renseigné, il ne manque plus qu'un certificat SSL pour pouvoir démarrer le serveur sécurisé. Voici la marche à suivre pour signer soi-même un certificat (il faut pour cela disposer d'OpenSSL, de toutes façons nécessaire pour installer **mod_ssl**).

PLUS LOIN **Certificat SSL**

SSL ne se contente pas de chiffrer le trafic, il assure également l'authentification du client et du serveur.
Celle-ci se fait par un certificat SSL, un document signé électroniquement par une autorité de certification. Il indique l'identité du serveur et la certifie exacte.
On peut évidemment se certifier soi-même en signant son propre certificat, mais les navigateurs n'acceptent par défaut que les autorités de certification bien connues, comme Verisign ou Thawte. En présence d'un certificat signé par une autre autorité, ils afficheront un message d'erreur alarmiste sur le fait qu'ils ont affaire à un certificat étrange et a priori peu digne de confiance.
Deux situations sont alors possibles :

- Sur un site Web sécurisé à vocation publique, destiné par exemple à du commerce électronique, il faudra passer sous les fourches caudines d'une autorité de certification connue, et s'acquitter annuellement de la somme qu'elle réclamera pour le renouvellement du certificat SSL.
- Sur un site à vocation interne à une institution, le choix est plus ouvert : louer un certificat ou créer une autorité de certification locale, qui signera elle-même les certificats. Elle devra alors être installée dans les navigateurs de tous les utilisateurs.

  Gratuite, l'auto-certification permet aussi de signer les certificats pour de longues périodes, sans devoir procéder à un renouvellement annuel.

Si l'authentification par certificat SSL est obligatoire sur les serveurs, elle n'est que rarement utilisée pour les clients.

```
$ export OPENSSL_CONF=/usr/share/examples/openssl/openssl.cnf ❶
$ mkdir -p demoCA/newcerts
$ touch demoCA/index.txt
$ cat > demoCA/serial
01
^D
$ openssl genrsa > exemple.key ❷
Generating RSA private key, 512 bit long modulus
..............++++++++++++
..++++++++++++
e is 65537 (0x10001)
$ openssl req -days 3650 -x509 -key exemple.key -new > cacert.pem ❸
Using configuration from /usr/share/examples/openssl/openssl.cnf
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:France
Locality Name (eg, city) []:Paris
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Ma Boite SARL
Organizational Unit Name (eg, section) []:Service Informatique
Common Name (eg, YOUR name) []:Ma Boite Certification Authority
Email Address []:root@example.com
$ openssl req -new -key exemple.key -out cert.csr ❹
Using configuration from /usr/share/examples/openssl/openssl.cnf
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:France
Locality Name (eg, city) []:Paris
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Ma Boite SARL
Organizational Unit Name (eg, section) []:Service Informatique
Common Name (eg, YOUR name) []:www.example.com
Email Address []:root@example.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
$ openssl ca -in cert.csr -keyfile exemple.key -cert cacert.pem -out cert.crt ❺
Using configuration from /usr/share/examples/openssl/openssl.cnf
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows
countryName           :PRINTABLE:'FR'
stateOrProvinceName   :PRINTABLE:'France'
localityName          :PRINTABLE:'Paris'
organizationName      :PRINTABLE:'Ma Boite SARL'
```

EN PRATIQUE **Le fichier openssl.cnf**

Son emplacement dépend un peu des systèmes. Par défaut, **openssl** le cherchera dans le répertoire /etc/openssl pour NetBSD, /etc/ssl pour FreeBSD.
NetBSD fournit un modèle pour ce fichier dans /usr/share/examples/openssl/openssl.cnf

VOCABULAIRE **Quelques abréviations**

Le monde de la cryptographie regorge lui aussi d'acronymes. Voici les plus courants en matière de SSL :

- SSL : *Secure Socket Layer* (couche sécurisée pour les connexions).
- CSR : *Certificate Signing Request* (requête de signature de certificat). Document transmis à l'autorité de certification.
- CA : *Certification Authority*. Autorité de certification.
- CRT : *CeRTificate*. Le certificat SSL à proprement parler.

```
organizationalUnitName:PRINTABLE:'Service Informatique'
commonName             :PRINTABLE:'www.example.com'
emailAddress           :IA5STRING:'root@example.com'
Certificate is to be certified until May 16 15:57:34 2004 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
$
```

❶ Préliminaires. Le fichier de configuration openssl.cnf, qui contient des réglages par défaut, sera placé dans le répertoire par défaut ou à l'endroit indiqué par la variable `OPENSSL_CONF`.

Une autorité de certification nécessite un répertoire (de nom précisé dans le fichier de configuration) contenant les fichiers serial et index.txt. Ce répertoire est inutile pour faire signer un certificat par un tiers.

❷ Génération de la clef privée qui va servir pour toutes ces opérations. C'est la clef de voûte de la sécurité des certificats : sa connaissance permettra à un tiers d'usurper votre identité.

❸ Création du certificat de l'autorité de certification. Les réponses aux questions posées seront consultables dans le certificat.

Cette étape est évidemment absente de la procédure de signature d'un certificat par une autre autorité de certification.

❹ Création de la requête de certification. On enverra ce fichier à l'éventuelle autorité externe chargée de le certifier, qui le retournera signé. Cette opération est décrite à l'étape suivante.

❺ Signature du certificat. Le fichier produit peut servir de certificat SSL pour le serveur Web.

Le fichier cert.crt ressemble à ceci :

```
-----BEGIN CERTIFICATE-----
MIIDUTCCAvugAwIBAgIBATANBgkqhkiG9w0BAQQFADCBoDELMAkGA1UEBhMCRlIx
DzANBgNVBAgTBkZyYW5jZTEOMAwGA1UEBxMFUGFyaXMxFjAUBgNVBAoTDU1hIEJv
aXRlIFNBUkwxHTAbBgNVBAsTFFNlcnZpY2UgSW5mb3JtYXRpcXVlMRgwFgYDVQQD
Ew93d3cubWFib2l0ZS5jb20xHzAdBgkqhkiG9w0BCQEWEHJvb3RAbWFib2l0ZS5j
b20wHhcNMDMwNTE3MTU1NzM0WhcNMDQwNTE2MTU1NzM0WjCBkDELMAkGA1UEBhMC
RlIxDzANBgNVBAgTBkZyYW5jZTEWMBQGA1UEChMNTWEgQm9pdGUgU0FSTDEdMBsG
A1UECxMUU2VydmljZSBJbmZvcmlhdGlxdWUxGDAWBgNVBAMTD3d3dy5tYWJvaXRl
LmNvbTEfMB0GCSqGSIb3DQEJARYQcm9vdEBtYWJvaXRlLmNvbTBcMA0GCSqGSIb3
DQEBAQUAA0sAMEgCQQDKhT7WB0Pr8baKv5UvIgabNWZQmBs1kX9SCKLXeht2UoaH
KnI6uJk969961KlfjZe7fxxUXljnPA665IDjAACZAgMBAAGjggEsMIIBKDAJBgNV
HRMEAjAAMCwGCWCGSAGG+EIBDQQfFh1PcGVuU1NMIEdlbmVyYXRlZCBDZXJ0aWZp
Y2F0ZTAdBgNVHQ4EFgQUkeaeZoitKRAy6atWajZmoP/v+YEwgc0GA1UdIwSBxTCB
woAUkeaeZoitKRAy6atWajZmoP/v+YGhgaakgaMwgaAxCzAJBgNVBAYTAkZSMQ8w
DQYDVQQIEwZGcmFuY2UxDjAMBgNVBAcTBVBhcmlzMRYwFAYDVQQKEw1NYSBCb210
ZSBTQVJMMR0wGwYDVQQLExRTZXJ2aWNlIEluZm9ybWF0aXF1ZTEYMBYGA1UEAxMP
d3d3Lm1hYm9pdGUuY29tMR8wHQYJKoZIhvcNAQkBFhByb290QG1hYm9pdGUuY29t
ggEAMA0GCSqGSIb3DQEBBAUAA0EAx124rqfvc/iMVGlbO6OTYP98/8vn2asNB+k0
0jUXBXp+5Tw9DSa4xvm5MkBuPlgI+GXZuM8LtKb/r0aAL3SeNw==
-----END CERTIFICATE-----
```

Il ne reste plus qu'à installer le certificat et la clef privée avec des permissions adéquates : seul root doit pouvoir les lire. La touche finale consiste à indiquer les emplacements de ces fichiers dans httpd.conf, et le serveur Web sera prêt à démarrer.

ALTERNATIVE **Script de création de certificats**

Le script Perl **`CA.pl`**, distribué avec OpenSSL, permet d'automatiser la génération des certificats.

SÉCURITÉ **Clef privée lisible par root uniquement ?**

La clef privée ne doit être lisible que par root. Ceci exclut donc le serveur Web lui-même, qui ne fonctionne normalement pas sous cet identifiant. Comment fait-il pour utiliser la clef ? C'est simple : il la lit au démarrage, avant de d'abandonner ses droits de root. Par la suite, le serveur se souvient de la clef, il n'a donc plus besoin de la lire.
Cette disposition permet de laisser les utilisateurs créer des scripts CGI, PHP, ou autres, sans leur laisser la possibilité de récupérer la clef. Si le serveur Web pouvait la lire, un simple script permettrait de la voler !

```
    SSLCertificateFile    /usr/local/private/cert.crt
    SSLCertificateKeyFile /usr/local/private/exemple.key
```

On le testera en invoquant un navigateur qu'on y branchera en précisant le protocole https:// en lieu et place du préfixe habituel http://. La commande **tcpdump** (probablement accompagnée des options **-s0 -X**) permettra de s'assurer que le trafic est chiffré – on prendra soin de se connecter aussi à un site non sécurisé pour disposer d'un point de comparaison.

Contenus dynamiques avec les CGI

CGI (*Common Gateway Interface*) est une spécification permettant à un serveur Web d'invoquer un programme externe pour servir une requête. Ce programme peut être écrit en n'importe quel langage : script shell, Perl, C, etc.

Faire fonctionner cette interface suppose :

- d' installer un serveur Apache avec le module **mod_cgi** ;
- d' indiquer dans httpd.conf quels types de fichiers seront considérés comme des CGI, en écrivant par exemple :

```
AddHandler cgi-script .cgi
```

- de s'assurer que les CGI sont bien exécutables : droits d'exécution sur le fichier, et pour tout script, ligne **#!** indiquant un interpréteur en état de fonctionnement.

Écrire un CGI

Voyons rapidement comment faire fonctionner un CGI. Soit une page HTML dotée d'un formulaire :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Mon test de CGI</title>
  </head>
  <body>
    <form method="POST" action="test.cgi">
      <table border="0">
        <tr>
          <td>Login: </td>
          <td><input type="text" name="login" value=""></td>
        </tr>
        <tr>
          <td>Mot de passe: </td>
          <td><input type="password" name="passwd" value=""></td>
        </tr>
        <tr>
          <td></td>
          <td align="right"><input type="submit" name="ok" value="Valider"></td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

La figure 12.1 en montre l'affichage par un navigateur Web.

SÉCURITÉ **Quel langage pour les CGI ?**

Le choix du langage à utiliser pour un CGI est assez sensible sur le plan de la sécurité. Ce programme manipulant des données fournies via le réseau par un utilisateur en général non authentifié, c'est une cible de choix pour les tentatives d'intrusion.

Le langage C crée des CGI très rapides qu'on peut facilement exécuter sous un identifiant choisi grâce à l'attribut *set-UID*. Hélas, de tels programmes seront plus enclins à contenir des erreurs de traitement des chaînes de caractères, compromettant la sécurité de l'édifice. Le C est donc un langage à réserver aux programmeurs qui savent ce qu'ils font lorsqu'il manipulent une chaîne de caractères.

Les scripts shell ne posent pas de tels problèmes de sécurité, mais les performances s'en ressentiront : chaque commande externe, comme **sed** ou **awk**, provoquera l'exécution d'un processus particulier.

Perl est un langage très apprécié pour les CGI, car il propose un bon compromis entre sécurité et performances. Ses capacités en traitement des chaînes de caractères faciliteront énormément la tâche du programmeur.

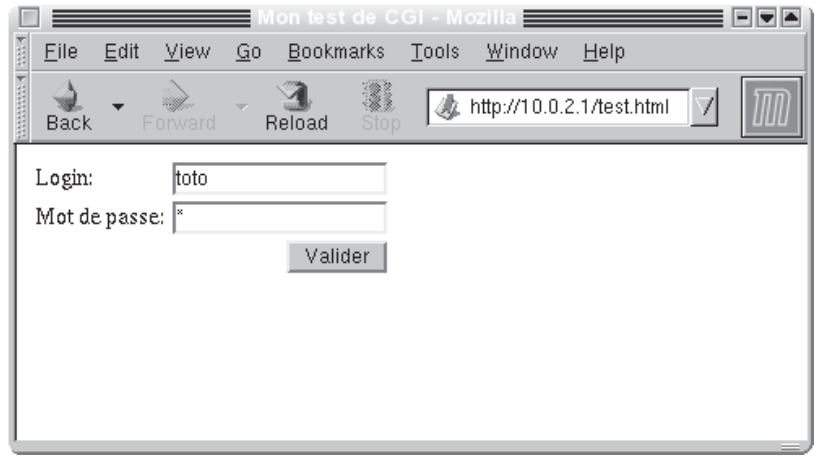


Figure 12–1 Test de CGI

Ce formulaire validé, le navigateur demande une URL désignant le programme test.cgi (placé ici dans le même répertoire que la page HTML). Le serveur reconnaîtra ce dernier comme un CGI et l'exécutera en lui passant un certain nombre de variables d'environnement :

SERVER_SOFTWARE	Nom et version du serveur Web. Exemple : `Apache/1.3.27`.
SERVER_NAME	Nom de la machine. Exemple : `www.example.com`.
GATEWAY_INTERFACE	Version de l'interface CGI. Exemple : `CGI/1.1`.
SERVER_PROTOCOL	Protocole utilisé par le serveur. Exemple : `HTTP/1.1`.
SERVER_PORT	Port TCP du serveur.
REQUEST_METHOD	Type de requête HTTP, le plus souvent `POST` ou `GET`.
SCRIPT_NAME	Chemin d'accès au CGI.
QUERY_STRING	Informations suivant le caractère `?` dans l'URL, lors de l'emploi de la méthode `GET`. Elles renferment les noms et valeurs des champs des formulaires.
REMOTE_HOST	Nom DNS du client.
REMOTE_ADDR	Adresse IP du client.
AUTH_TYPE	Type d'authentification utilisée, le cas échéant.
REMOTE_USER	En cas d'authentification, nom de l'utilisateur.
CONTENT_TYPE	Dans les requêtes où le client envoie un document (le plus souvent `POST`), type MIME du document. Lors du traitement d'un formulaire utilisant la méthode `POST`, les noms et valeurs des champs du formulaire sont fournis au serveur dans un document texte (de type `text/plain` le plus souvent).
CONTENT_LENGTH	Taille du document transmis par le client.

CULTURE **Types MIME**

Les types MIME (*Multipurpose Internet Mail Extensions*) sont des brèves chaînes de caractères décrivant le format d'un document. Exemple de type MIME pour décrire un document HTML : `text/html`.
Ces types ont été introduits initialement dans la messagerie (d'où leur nom), pour permettre l'envoi de pièces jointes et de documents dans d'autres formats que le texte brut. Des services tels que le Web utilisent abondamment les types MIME pour indiquer la nature des données échangées entre le client et le serveur.

Les noms et valeurs des champs du formulaire sont fournis au serveur sous la forme d'une chaîne telle que :

```
login=toto&passwd=q
```

Cette chaîne est transmise au CGI dans la variable d'environnement `QUERY_STRING` pour les formulaires utilisant la méthode `GET`, et sur l'entrée standard du CGI pour ceux qui recourent à la méthode `POST`.

Voici un exemple de script Perl permettant d'afficher le informations transmises par le formulaire présenté précédemment :

```
#!/usr/local/bin/perl ❶

sub getparam {
    local(%postdata) = ();
    if($ENV{'REQUEST_METHOD'} eq 'POST') { ❷
        local($len) = $ENV{'CONTENT_LENGTH'}; ❸
        local($data) = "";
        if(read(STDIN, $data, $len) != $len) { ❹
            print "<h1>Erreur de lecture sur les donn&eacute;es POST</h1>\n";
            print "</body></html>\n";
            die("Erreur de lecture sur les donn&eacute;es POST\n"); ❺
        }
        foreach $d (split('&', $data)) { ❻
            local($nom, $valeur) = split('=', $d); ❼
            $postdata{$nom} = $valeur;
        }
    } else {
        print "<h1>Erreur: le formulaire doit &ecirc;tre de type POST</h1>\n";
        print "</body></html>\n";
        die("Erreur: le formulaire doit &ecirc;tre de type POST\n");
    }
    return %postdata;
}

print "Content-type: text/html", "\n\n"; ❽
print "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\">\n\n";
print "<html><head><title>R&eacute;sultat du formulaire</title>\n";
print "<meta http-equiv=\"Content-Type\"\n";
print "      content=\"text/html; charset=iso-8859-1\"></head><body>\n";

%form=&getparam;

print "<table border=\"1\">\n";
print "<tr><th>Champ</th><th>Valeur</th></tr>\n";
foreach $key (sort keys(%form)) { ❾
    print "<tr><td>$key</td><td>$form{$key}</td></tr>\n";
}
print "</table>\n";

print "</body></html>\n";
__END__
```

❶ Le nom de l'interpréteur du script, le cas échéant. Cette ligne, obligatoire, doit indiquer le chemin d'un interpréteur fonctionnel, sans quoi le CGI ne pourra pas s'exécuter. Cette obligation ne s'applique évidemment pas aux sources des CGI écrits dans des langages compilés comme le C.

❷ Ce script ne gère que les requêtes POST. Il vérifie donc que la variable d'environnement `REQUEST_METHOD` indique bien `POST`. En Perl, on utilise le tableau associatif `ENV` pour obtenir les valeurs des variables d'environnement.

❸ La taille des données transmises par le client est donnée par la variable d'environnement `CONTENT_LENGTH`.

❹ Dans une requête POST, les données sont transmises du serveur Web au CGI par l'entrée standard (*stdin*). On stocke son contenu dans `$data`, et on en

ALTERNATIVE `POST` ou `GET` ?

Quelle méthode choisir pour les formulaires ?
La taille des informations transmises avec la méthode `GET` est souvent limitée par le navigateur (exemple : 2083 octets pour Internet Explorer au moins jusqu'à la version 5.5). De plus, la chaîne de requête apparaît dans la nouvelle URL, ce qu'on évitera évidemment pour transmettre un mot de passe. C'est la méthode à retenir pour les contenus dynamiques évoluant lentement et à vocation consultative, car on pourra noter l'URL résultat du formulaire dans des signets, et la page Web ainsi produite pourra être stockée dans des serveurs mandataires.
`POST` n'a pas de limite de taille, et n'affiche à l'écran aucune donnée issue du formulaire – on la préférera donc lors de la transmission d'informations privées. C'est aussi la méthode à utiliser pour les contenus dynamiques à faible durée de validité.

ALTERNATIVE CGI.pm

Cet exemple est surtout intéressant pour comprendre comment les CGI fonctionnent. On évite en général de reprogrammer systématiquement les traitements nécessaires à la récupération des valeurs du formulaire, le module Perl CGI.pm pouvant s'en charger.
Pour plus d'informations sur CGI.pm, reportez-vous à sa documentation : http://stein.cshl.org/WWW/software/CGI

EN CAS DE COUP DUR **Si le CGI ne fonctionne pas**

- Vérifiez que le script est bien lisible et exécutable pour l'UID du serveur Web. Dans le doute, mettez-le en mode 755 ainsi que tous les répertoires qui y mènent.
- Assurez-vous que l'interpréteur précisé dans la première ligne est correct. Le CGI doit pouvoir s'exécuter à la ligne de commande.
- En l'absence de message d'erreur, si le client reçoit un document vide, assurez-vous d'avoir fait afficher les en-têtes HTTP par le CGI.
- Assurez-vous, dans la configuration d'Apache, que le répertoire dispose des permissions **ExecCGI**. L'option **XBitHack** permet de considérer les fichiers exécutables comme des CGI.
- Si le CGI s'exécute mais produit une erreur interne, vérifiez les journaux d'erreur d'Apache : la raison du problème devrait y figurer. De tels problèmes peuvent être produits par une erreur de syntaxe.

Pour tester les CGI en ligne de commande, employez **su** pour prendre l'UID de l'utilisateur sous l'identité duquel fonctionne Apache, et exécutez le CGI en lui fournissant l'environnement et l'entrée standard qu'Apache lui aurait fournis. Exemple :

```
# su www
$ export QUERY_STRING="login=toto&passwd=q"
$ export REQUEST_METHOD="POST"
$ CONTENT_LENGTH=`echo $QUERY_STRING|wc|awk '{print $3}'`
$ export CONTENT_LENGTH
$ echo $QUERY_STRING|./test.cgi
Content-type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN">

<html><head><title>Résultat du formulaire</title>
<meta http-equiv="Content-Type"
      content="text/html; charset=iso-8859-1"></head><body>
<table border="1">
<tr><th>Champ</th><th>Valeur</th></tr>
<tr><td>login</td><td>toto</td></tr>
<tr><td>passwd</td><td>q
</td></tr>
</table>
</body></html>
```

profite pour contrôler la bonne réception de toutes les données attendues. Dans le cas contraire, c'est une erreur.

❺ Les messages d'erreur passés à `die()` ne sont pas envoyés au client : l'administrateur peut les consulter dans les journaux d'erreur d'Apache.

❻ C'est ici que l'on sépare la chaîne `login=toto&passwd=q` en ses éléments constitutifs `login=toto` et `passwd=q`.

❼ Découpages des chaînes `login=toto`, et `passwd=q` en couples (nom, valeur).

❽ Quand le serveur Web exécute un CGI, il ne gère pas les en-têtes HTTP, que le CGI doit donc produire lui-même. Elles sont séparées du reste du document par une ligne vide, ce qui explique le double retour chariot.

❾ Il ne reste plus qu'à afficher le résultat. Dans la pratique, ils seront probablement exploités autrement, mais ce script donne la structure qui vous mettra le pied à l'étrier.

Le résultat dans le navigateur du client est visible dans la figure 12.2.

Contenus dynamiques avec PHP

Voyons rapidement la mise en place de contenus dynamiques avec PHP. Ils nécessitent de compiler Apache avec **mod_php**, de configurer PHP (par le biais du fichier php.ini), voire d'installer des modules pour PHP en cas de besoins particuliers.

En passant par un système de paquetages, tout cela est assez simple. Sur NetBSD par exemple, il suffira d'installer apache et ap-php4 de la catégorie www, et si par exemple vous désirez générer des images dynamiques en PHP, le paquetage php4-gd de la catégorie graphics.

PLUS LOIN **Modules PHP**

PHP a la même structure modulaire qu'Apache. Certaines de ses fonctionnalités se trouvent dans des modules, qui sont éventuellement dynamiques. C'est le cas par exemple des modules d'interfaçage avec les bases de données comme MySQL et PostgreSQL.

Bien qu'ils fonctionnent sur le même principe, ces modules ne sont pas interchangeables avec ceux d'Apache.

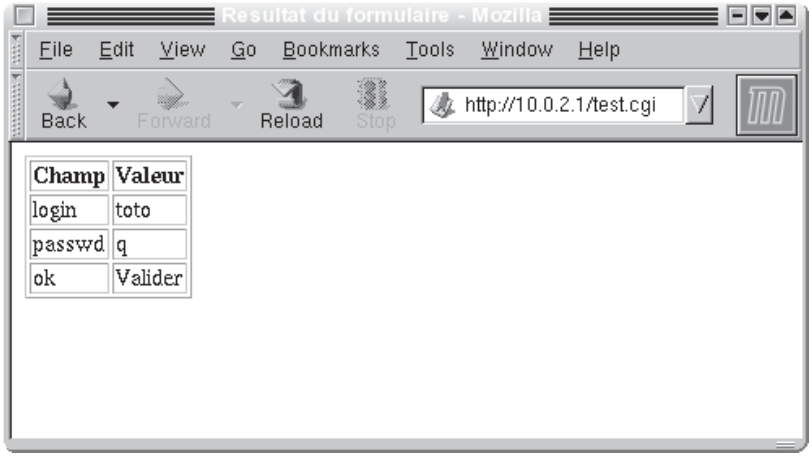


Figure 12–2 Résultat du CGI

Reste ensuite un peu de configuration :

- À moins que vous n'ayez compilé vous-même Apache avec un **`mod_php4`** statique, il faut demander le chargement de ce dernier dans httpd.conf : **`LoadModule php4_module /usr/pkg/lib/httpd/mod_php4.so`**.
- Toujours dans httpd.conf, il faut indiquer que les fichiers de suffixe .php sont à traiter par PHP : **`AddType application/x-httpd-php .php`** (vous pouvez faire de même pour les fichiers .html, mais cela a un coût en termes de performances : toutes les pages HTML servies passeront par le moteur PHP).
- Indiquer dans php.ini les modules PHP dynamiques que l'on désire charger par une ligne comme **`extension=gd.so`**.
- Apache est enfin prêt à redémarrer.

Voici un script de test pour vérifier que PHP fonctionne bien :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    <title>Mon test en PHP</title>
  </head>
  <body>
    <?php echo "Hello world"; ?>
  </body>
</html>
```

Si `Hello world` s'affiche, alors PHP fonctionne. Sinon, contrôlez à nouveau la configuration et plongez dans les journaux (en particulier error_log) pour constater toute erreur.

SÉCURITÉ **Qui réclame mon `command.com` ?**

Puisqu'on mentionne les journaux d'erreur... toute connexion à Internet y laissera des traces de requêtes délirantes, telles que des tentatives d'accès à command.com (équivalent du shell sous MS-DOS, sans aucun lien avec Apache et Unix).

Ce sont des tentatives d'intrusion qui ne tiennent pas compte du système d'exploitation utilisé. D'autres essais cibleront Unix. Ces coups de boutoir sont la raison pour laquelle il faut régulièrement mettre à jour l'installation : tant que les assaillants tentent d'exploiter des vulnérabilités que vous avez corrigées, ils ne pourront pas s'introduire dans votre système.

Serveur DNS : fonctionnement et mise en œuvre

Sujet largement abordé dans les chapitres précédents, les serveurs DNS ne demandent plus qu'à être traités de front.

Fonctionnement

Examinons une configuration renfermant presque toutes les situations possibles. Un réseau a pour nom de domaine `example.net` et contient notamment quatre machines :

- `sinusite.rd.example.net` (`192.0.2.15`), station de travail normale, jouera le rôle du client. Elle est configurée pour utiliser le serveur de noms `dns.rd.example.net`.
- `dns.rd.example.net` (`192.0.2.1`), serveur de noms primaire pour le domaine `rd.example.net`. Pour tout le reste, il interroge `ns.example.net`.
- `www.rd.example.net` (`192.0.2.2`), serveur Web.
- `ns.example.net` (`192.0.2.153`), serveur primaire du domaine `example.net`.
- `www.netbsd.org` (`204.152.184.116`), serveur sur Internet.

Une requête simple

La figure 12.3 décrit la situation. `sinusite.rd.example.net` tente une connexion Web à `www.rd.example.net` : il lui faut donc l'adresse IP de cette machine. Elle interroge pour cela son DNS, `dns.rd.example.net`, et lui demande la classe `IN`, type `A`, pour `www.rd.example.net`.

`dns.rd.example.net` est serveur primaire pour la zone `rd.example.net`, il fait donc autorité et n'a rien à demander à personne. Il consulte son fichier de zone pour `rd.example.net` et envoie la réponse à `sinusite` : `192.0.2.2`.

CULTURE **Les autres classes du DNS**

Par défaut, **`dig`** et **`nslookup`** font leurs requêtes dans la classe `IN`. Les autres classes sont d'un usage beaucoup plus rare :

- `HESIOD` (ou `HS`) est utilisé par le système d'annuaire Hesiod, développé au *Massachusetts Institute of Technology* (MIT) dans le cadre du projet Athena. Du point de vue des fonctionnalités, Hesiod est semblable à NIS.
- `CHAOS` (ou `CN`) provient d'un protocole de communication pour réseaux locaux appelé CHAOSnet. Il s'agit d'un autre développement du MIT, aujourd'hui tombé en désuétude.

Référez-vous à la documentation si vous devez indiquer à **`dig`** ou **`nslookup`** une autre classe que `IN` – mais il est peu probable que vous en éprouviez jamais le besoin.

VOCABULAIRE **Fichier de zone**

Le fichier de zone est un fichier texte renfermant tout ce que le serveur de noms doit savoir sur un domaine. On y trouve des lignes comme :

```
sinusite  IN  A  192.0.2.15
www       IN  A  192.0.2.2
```

Le fichier de zone est mis à jour par l'administrateur du serveur de noms.

RAPPEL **Classes et types du DNS**

Nous avons déjà évoqué le DNS au chapitre 9. Chaque requête concerne une classe (presque toujours `IN` comme *INternet*), un type et une adresse DNS. Exemples de types :

`A`	Adresse IP correspondant à une adresse DNS.
`PTR`	Adresse DNS correspondant à une adresse IP. L'adresse IP est « retournée » et incluse dans le domaine in-addr.arpa. Pour obtenir par exemple l'adresse DNS correspondant à `10.1.12.4`, on demande la classe `IN`, type `PTR` pour l'adresse `4.12.1.10.in-addr.arpa`.
`NS`	Les serveurs de noms faisant autorité pour un nom de domaine DNS.
`MX`	Les serveurs de messagerie recevant le courrier pour un domaine DNS.
`CNAME`	Les alias d'une adresse DNS.

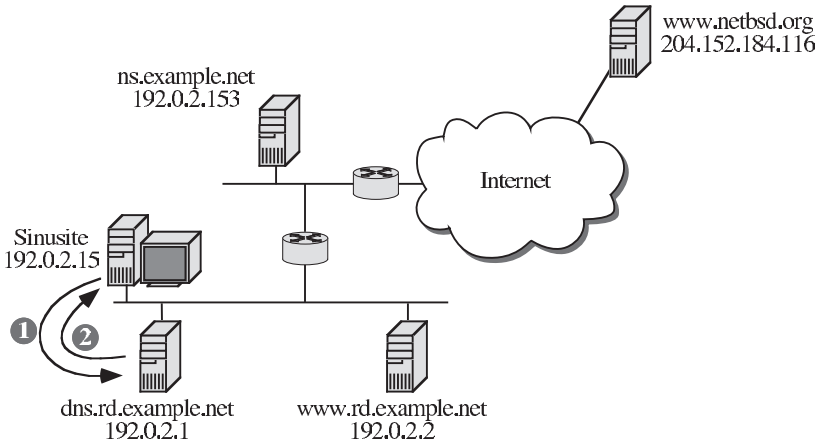


Figure 12–3 Une requête DNS simple

Requêtes à faire suivre

Si l'utilisateur de `sinusite` désire connaître l'adresse IP du DNS primaire du domaine `example.net`, il fait une requête de classe `IN`, type `NS` avec le nom `example.net`. On peut même opérer en ligne de commande :

```
$ host -t NS example.net
```

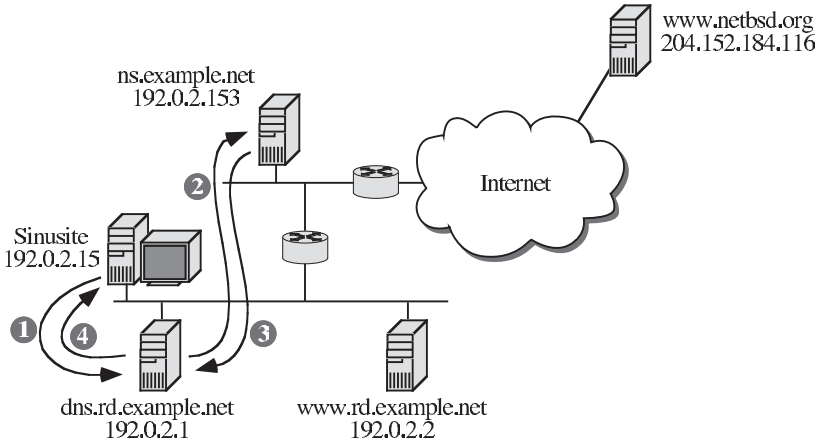


Figure 12–4 Requête DNS à faire suivre

ALTERNATIVE **host, dig, et nslookup**

Ces trois commandes permettent d'extraire des informations semblables. **`nslookup`** est une commande historique, vouée à être remplacée par **`host`** et **`dig`**. Sa syntaxe est différente de celle de ses deux remplaçants : pour obtenir un type `NS`, on écrit par exemple **`nslookup -query=NS example.net`**, **`dig -t NS example.net`**, ou **`host -t NS example.net`**.

La figure 12.4 détaille cette procédure. `sinusite` envoie la requête à son serveur de noms, `dns.rd.example.net`. Celui ci ne fait autorité que pour `rd.example.net`, et ne sait rien du reste de la zone `example.net`. Il est configuré pour faire suivre à `ns.example.net` toutes les requêtes qu'il ne peut traiter lui-même. `ns.example.net` fait autorité pour le domaine `example.net`, puis-

qu'il en est le serveur primaire. Il consulte son fichier de zone, répond à `dns.rd.example.net`, qui renvoie à son tour la réponse à `sinusite.rd.example.net`.

Ce comportement consistant à faire suivre les requêtes puis leurs réponses s'appelle en anglais une configuration en *forwarder*.

PERFORMANCES **Requêtes DNS**

Dans le cas qui nous intéresse, `dns.rd.example.net` fait autorité pour au moins une zone, mais on configure parfois des serveurs de noms qui font suivre toutes les requêtes. Ces configurations ont un intérêt dans la mesure où les serveurs de noms sont capables de stocker des résultats en « cache » (antémémoire). On peut ainsi optimiser le trafic entre une partie du réseau et le serveur primaire de l'entreprise en utilisant sur celle-là un DNS qui se contente de faire du cache.

Requêtes récursives

EN PRATIQUE **Requêtes récursives**

Dans cet exemple, la requête récursive s'arrête à `dns.rd.example.net`, mais on peut très bien obtenir des chaînes de récursion plus longues, en particulier en présence d'un plus grand nombre de segments dans le nom DNS.

Pour une raison étrange, `www.rd.example.net` est configuré pour utiliser le DNS primaire de la société : `ns.example.net`. S'il doit résoudre l'adresse `sinusite.rd.example.net`, il envoie à son serveur de noms la requête de classe `IN`, type `A`, et nom `sinusite.rd.example.net`, opération détaillée sur la figure 12.5.

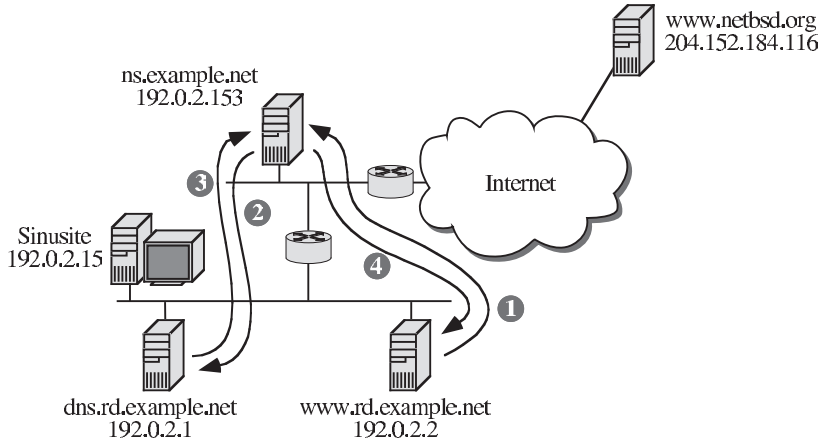


Figure 12–5 Requête DNS récursive

`ns.example.net` ne connaît pas la réponse car il ne fait pas autorité pour `rd.example.net`,. Cependant, il connaît la machine qui fait autorité : `dns.rd.example.net` ; il lui transmet donc cette requête.

`dns.rd.example.net` fait autorité pour `rd.example.net` ; il peut donc donner la réponse (`192.0.2.15`), qu'il renvoie à `ns.example.net`, lequel transmet à `www.rd.example.net`.

Ce type de comportement est une requête récursive : un serveur ne connaît pas la réponse, mais connaît le serveur qui connaît la réponse. Chacun fait suivre, et la réponse prend le chemin inverse.

> PLUS LOIN **Requêtes à faire suivre et requêtes récursives**
>
> La différence entre les requêtes à faire suivre et les requêtes récursives est subtile. Dans le cas des requêtes à faire suivre, un DNS est configuré pour transmettre toutes les requêtes pour lesquelles il ne fait pas autorité à un autre serveur, explicitement indiqué dans la configuration.
> Dans le cas des requêtes récursives, la requête est transmise de serveur en serveur, mais le serveur contacté n'est pas toujours le même : la hiérarchie du DNS dicte le serveur qui fait autorité.

Requêtes sur les serveurs racine

Que se passe-t-il quand le serveur ne connaît pas la machine qui fait autorité ? Ce cas de figure est représenté figure 12.6. Si `sinusite.rd.example.net` souhaite contacter `www.netbsd.org`, elle a besoin de son adresse IP. Elle envoie donc à son DNS `dns.rd.example.net` une requête de classe `IN`, type `A`, adresse DNS `www.netbsd.org`.

`dns.rd.example.net` ne connaît pas la réponse ; il fait suivre à `ns.example.net`, puisqu'il est configuré comme *forwarder*. `ns.example.net` ne connaît pas non plus la réponse, mais ne sait pas à qui faire suivre. Cette réponse est disponible sur le serveur faisant autorité pour le domaine `netbsd.org`, mais on ne le connaît pas. C'est le moment de recourir aux serveurs racine.

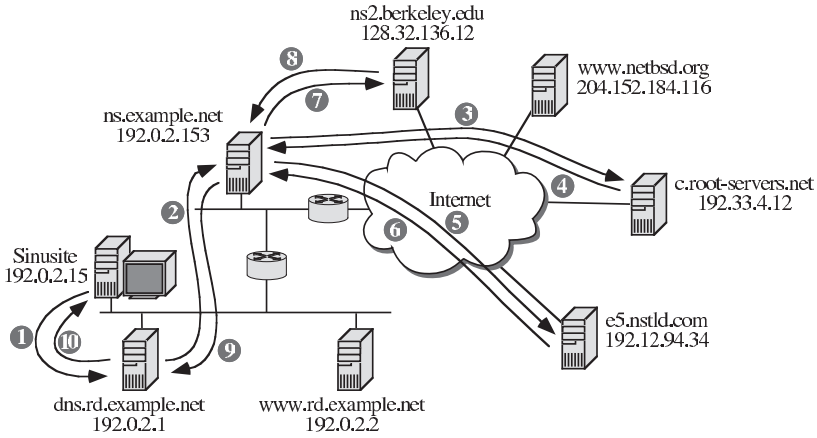


Figure 12–6 Requête DNS à la racine

Dans sa configuration, le serveur de noms connaît en effet une liste de serveurs dits racine. Ils connaissent les adresses des serveurs qui font autorité pour tous les domaines du premier niveau de la hiérarchie : `.com`, `.org`, `.int`, `.fr`, `.de`, etc.

EN PRATIQUE **Serveur qui fait autorité pour `.org` ?**

La liste des serveurs qui font autorité pour `.org` ne sera pas transmise jusqu'au client qui a émis la requête, mais les curieux peuvent la demander explicitement :

```
$ dig -t NS org
; <<>> DiG 8.3 <<>> ns org
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2
;; flags: qr rd ra; QUERY: 1, ANSWER: 9, AUTHORITY: 0, ADDITIONAL: 0
;; QUERY SECTION:
;;      org, type = NS, class = IN

;; ANSWER SECTION:
org.                    6D IN NS        G7.NSTLD.COM.
org.                    6D IN NS        I5.NSTLD.COM.
org.                    6D IN NS        J5.NSTLD.COM.
org.                    6D IN NS        L7.NSTLD.COM.
org.                    6D IN NS        M5.NSTLD.COM.
org.                    6D IN NS        A7.NSTLD.COM.
org.                    6D IN NS        C5.NSTLD.COM.
org.                    6D IN NS        E5.NSTLD.COM.
org.                    6D IN NS        F7.NSTLD.COM.

;; Total query time: 103 msec
;; FROM: mafate.sis.pasteur.fr to SERVER: default -- 157.99.64.64
;; WHEN: Tue Jul 15 11:20:52 2003
;; MSG SIZE  sent: 21  rcvd: 183
```

`ns.example.net` envoie donc la requête à un serveur racine, par exemple `c.root-servers.net`. Ce serveur ne connaît pas le serveur qui fait autorité pour `netbsd.org`, mais il connaît ceux de `.org`. La requête pourrait se faire récursivement, du serveur racine à un serveur qui fait autorité pour `.org`, puis au serveur qui fait autorité pour `netbsd.org`, mais il n'en sera rien : `c.root-servers.net` est un serveur très occupé, il se contentera donc de fournir à `ns.example.net` la liste des serveurs qui font autorité pour le domaine `.org`.

`ns.example.net` pourra maintenant interroger un serveur qui fait autorité pour `.org`. Il transmet la requête à l'un d'entre eux, choisi au hasard dans la liste renvoyée par `c.root-servers.net` : `e5.nstld.com`. Ce serveur, lui aussi très occupé, ne fait pas de récursion, mais se contente de renvoyer la liste de serveurs qui font autorité pour `netbsd.org`.

`ns.example.net` pourra désormais interroger l'un des serveurs qui font autorité pour le domaine `netbsd.org`. Au hasard, c'est `ns2.berkeley.edu` qui fournira enfin la réponse tant convoitée : `204.152.184.116`, renvoyée à `dns.rd.example.net` d'où elle rebondit sur `sinusite.rd.example.net`. La boucle est bouclée.

À lire le déroulement de la résolution, on peut craindre pour les performances, mais n'oublions pas que chaque serveur cache les réponses. Ainsi, la réponse de la requête (`IN`, `A`, `www.netbsd.org`) sera retenue par `dns.rd.example.net` et `ns.example.net`, mais `ns.example.net` se souviendra aussi quelque temps des adresses des serveurs faisant autorité pour les domaines `netbsd.org` et `.org`.

EN PRATIQUE **Serveur qui fait autorité pour `netbsd.org` ?**

On découvrira les serveurs qui font autorité pour `netbsd.org` de la même manière qu'on a obtenu ceux du domaine `.org` :

```
$ host -t NS netbsd.org
netbsd.org name server ns2.berkeley.edu
netbsd.org name server adns1.berkeley.edu
netbsd.org name server adns2.berkeley.edu
netbsd.org name server uucp-gw-1.pa.dec.com
netbsd.org name server uucp-gw-2.pa.dec.com
netbsd.org name server ns.netbsd.org
netbsd.org name server ns1.berkeley.edu
```

Configuration de named

L'implémentation de très loin la plus répandue du serveur DNS est BIND (*Berkeley Internet Name Domain*), aujourd'hui développé par l'*Internet Software Consortium* (ISC) et intégrée à de très nombreux systèmes, dont les BSD. Ce logiciel inclut un certain nombre de programmes, dont le serveur DNS lui-même : **named**.

named utilise le fichier de configuration named.conf, où l'on indique pour quelles zones le serveur sera DNS primaire ou secondaire. On y précise aussi dans quelles circonstances le serveur doit accepter les requêtes récursives ou se comporter en *forwarder*. Enfin, le fichier named.conf donne à **named** les emplacements des fichiers de zones pour lesquelles il fera autorité, et l'emplacement du fichier db.cache, contenant la liste des serveurs racine.

La lecture de la page **man** de named.conf est indispensable à la configuration de named. Sur NetBSD et FreeBSD, un fichier de configuration d'exemple se trouve dans /etc/namedb/named.conf. En bref, il comporte une section d'options générales, suivie d'une section par zone.

Exemple de déclaration d'une zone pour laquelle le serveur est DNS primaire (zone `example.net` ; fichier de zone exemple) :

```
zone "example.net" {
   type master;
   file "exemple";
};
```

SÉCURITÉ **named**

Tout comme **httpd**, **named** peut fonctionner sans les droits de root, possibilité qu'il est fortement recommandé d'exploiter.
Les options **-u** et **-g** permettent de choisir l'utilisateur et le groupe du processus **named**.

ALTERNATIVE **Vieux named**

On compte trois branches de développement de BIND : BIND4, BIND8, et BIND9. Dans BIND4, le fichier de configuration a une syntaxe assez différente et s'appelle named.boot. BIND4 et BIND8 sont des bases de code anciennes et moins sécurisées, et l'ISC recommande désormais BIND9.

EN PRATIQUE **Les serveurs racine**

Le liste des serveurs racine se trouve à l'adresse ftp://ftp.rs.internic.net/domain/db.cache. Ce répertoire contient également le fichier de la zone racine, root.zone, qu'ils utilisent.

IMPORTANT **Zone racine**

N'oubliez pas d'indiquer le fichier contenant les serveurs racine, faute de quoi le serveur DNS ne pourra pas résoudre les adresses pour lesquelles il ne fait pas autorité.

```
zone "." {
   type hint;
   file "db.cache";
};
```

Configuration en DNS primaire

Le serveur primaire est l'un de ceux qui font autorité pour une zone. C'est là que le fichier de zone est mis à jour ; il sera ensuite recopié sur les serveurs secondaires.

Vus de l'extérieur, les DNS primaire et secondaires ont un rôle parfaitement symétrique : ils sont tous mentionnés dans les fichiers de zone pour le type `NS`, et rien ne les distingue. Enfin, presque rien : en surveillant les changements qui concernent un domaine sur tous ses DNS qui font autorité, on peut remarquer que l'un d'entre eux dispose en premier des modifications : c'est le DNS primaire.

Pour l'administrateur, la différence entre le primaire et les secondaires est importante. La configuration d'un DNS secondaire se limite à indiquer qu'il est secondaire pour telle zone et que le primaire correspondant se trouve à telle adresse. Le DNS secondaire s'y connectera régulièrement pour obtenir les éventuelles mises à jour du fichier de zone (on parle de « transfert de zone »).

SÉCURITÉ **Redondance du DNS**

Pour assurer une bonne résistance aux pannes, chaque domaine DNS doit compter au moins un DNS secondaire, mais il est possible d'en avoir plusieurs. Pour la même raison, il est recommandé de ne pas tous les placer sur le même réseau.

Sur le DNS primaire, on trouve la version maître du fichier de zone, que l'administrateur doit tenir à jour. En voici un exemple :

```
$TTL 518400 ❶
@         IN    SOA     ns.example.net. hostmaster.example.net. ( ❷
                                  2003051700 ;serial
                                  1800 ;refresh every 30 min
                                  900 ;retry every 15 min
                                  604800 ;expire after a week
                                  86400 ;minimum of a day
                                  )
          IN    NS      ns.example.net. ❸
          IN    NS      ns2.example.net.
          IN    NS      ns1.nic.fr.

rd        IN    NS      dns.rd.example.net. ❹

          IN    MX 10   minas-morgul.example.net. ❺
          IN    MX 20   khazad-dum.example.net.

ns        IN    A       192.0.2.153 ❻
ns2       IN    A       192.0.2.129
khazad-dum      IN    A       192.0.2.88
minas-morgul    IN    A       192.0.2.100

www       IN    CNAME   khazad-dum.example.net. ❼
mail      IN    CNAME   minas-morgul.example.net.
(...)
```

ATTENTION Le point de fin de ligne
Les points de fin de ligne sont importants : ils évitent aux réponses d'être suffixées par le nom de domaine. Sans eux, les résolutions d'adresses IP donneraient par exemple `ns.example.net.example.net`.

❶ La directive TTL indique à **named** la durée de validité par défaut, exprimée en secondes, pour toutes les lignes du fichier de zone. On peut indiquer un TTL différent pour une ligne en précédant `IN` de son TTL. Ces durées de validité affectent les caches des serveurs de noms.

❷ Le type SOA (*Start Of Authority*) est toujours le premier dans un fichier de zone. Il indique l'adresse du DNS primaire, celle de son administrateur (où le caractère arobase `@` est remplacé par un point `.`), puis, entre parenthèses, les informations déterminant le rythme des transferts de zone par les DNS secondaires.

L'information la plus importante est sans conteste le numéro de série (`serial`), qu'il faut incrémenter à chaque modification du fichier de zone. C'est

PLUS LOIN zone restreintes

Outre le faire fonctionner sous un UID non root, il existe un autre mécanisme pour augmenter la sécurité de **named** : le faire fonctionner dans une zone restreinte, grâce à l'option **-t**.
Les zones restreintes (en jargon, on parle de zone « chrootée », ce qui se prononce zone « c-h-routée ») permettent d'enfermer un processus dans une arborescence restreinte du système de fichiers. Le processus voit le sommet de l'arborescence restreinte comme racine. Exemple de construction de zone restreinte :

```
# mkdir -p /tmp/chroot/bin
# cp /bin/sh /bin/ls /tmp/chroot/bin
# SHELL=/bin/sh
# chroot /tmp/chroot
# ls /
bin
# ls /bin
ls sh
```

Attention : si sur votre système **sh** et **ls** sont des binaires dynamiques, alors vous ne pourrez pas les exécuter dans l'arborescence restreinte sans y copier aussi les fichiers nécessaires à la liaison dynamique : l'éditeur de liens dynamiques et les bibliothèques avec lesquelles les programmes sont liés.
La raison d'être des zones restreintes est la garantie d'un plus grand niveau de sécurité : si un assaillant réussit à prendre le contrôle d'un *daemon* fonctionnant en arborescence restreinte, il ne gagnera un accès qu'à cette arborescence. Pour accéder à l'ensemble du système, il faudra trouver un autre trou de sécurité, dans le noyau lui-même.
Pour plus d'informations sur les zones restreintes, vous pouvez consulter un article écrit par l'auteur pour ONLamp.com à l'adresse suivante : http://www.onlamp.com/pub/a/bsd/2003/01/23/chroot.html.

ce qui permet aux DNS secondaires de constater que le fichier de zone a changé. En l'absence d'incrémentation, aucun transfert de zone n'aura lieu.

❸ Les types `NS` indiquent les serveurs de noms qui font autorité pour la zone. Les points de fin de ligne sont indispensables, comme expliqué dans l'encadré.

❹ Exemple de délégation d'un sous-domaine à un autre serveur DNS.

❺ Le type `MX` indique les serveurs de messagerie recevant le courrier pour le domaine. Nous y reviendrons dans la section consacrée à la messagerie.

❻ Les types `A` définissent des adresses IP.

❼ Les types `CNAME` définissent des alias DNS.

Toute modification d'un fichier de zone sera immédiatement prise en compte par **`named`** si on lui envoie un signal `SIGHUP` (**`kill -1`**).

PIÈGE À C... **Chargement incorrect des zones**

Lors du rechargement des fichiers de zones, n'oubliez pas de contrôler dans les journaux du système (en général /var/log/messages) que **`named`** a correctement rechargé toutes les zones. En cas d'erreur de syntaxe dans l'un des fichiers, **`named`** ignorera la zone et passera à la suite. Le serveur cessera alors de servir les requêtes relatives au domaine concerné.

PIÈGE À C... **Transfert de zone**

Les transactions DNS, habituellement en UDP sur le port 53, se font en TCP pour les transferts de zone du DNS primaire à un DNS secondaire. Ce dernier ouvre alors une connexion TCP sur le port 53 du DNS primaire.

Si un DNS primaire est placé derrière un dispositif de filtrage, il faudra donc penser à autoriser les DNS secondaires à se connecter sur son port TCP 53.

La commande **`named-xfer`** permet de tester les transferts de zone. Sur les BSD, elle se trouve dans /usr/libexec.

Autre exception, plus rare : quand la réponse à une requête dépasse 512 octets, elle repartira avec le bit de troncature activé ce qui obligera le client à refaire sa requête en TCP.

Le DNS inverse

La fonction première du DNS est de traduire des adresses DNS en adresses IP, mais il est aussi utilisé pour l'opération inverse. C'est ce qu'on appelle le *reverse DNS*, ou DNS inverse.

Comme on l'a vu, les requêtes de DNS inverse se font en demandant un type **`PTR`** dans le domaine in-addr.arpa. Les adresses IP s'écrivent à l'envers, ce qui peut surprendre au début.

Pour mettre en place le DNS inverse de la plage `192.168`, il faut se déclarer le fichier named.conf serveur primaire du domaine `168.192.in-addr.arpa` :

```
zone "168.192.in-addr.arpa" {
   type master;
   file "192.168";
};
```

Le fichier de zone contiendra des champs tels que :

```
1.0   IN  PTR  diplodocus.chezmoi.local.
2.0   IN  PTR  barbapapa.chezmoi.local.
```

On vérifie ensuite que tout fonctionne correctement :

```
$ host 192.168.0.1
1.0.168.192.IN-ADDR.ARPA domain name pointer diplodocus.chezmoi.local
```

PIÈGE À C... **Numéro de série du fichier de zone**

Il est *primordial* de penser à incrémenter le numéro de série du fichier de zone à chaque modification. À défaut, les DNS secondaires ne feront pas de transfert de zone, et ils conserveront une ancienne version du fichier. Conséquence : selon que les clients interrogeront le DNS primaire ou un DNS secondaire, ils obtiendront des résultats différents.

RAPPEL

Avez-vous pensé à incrémenter le numéro de série du fichier de zone ?

Le meilleur et le pire de la messagerie

La messagerie est le service le plus complexe à configurer. Les cas particuliers à surveiller abondent, sans aucun droit à l'erreur : le courrier des utilisateurs est en effet la ressource la plus sensible d'un système informatique. Qui n'en est pas convaincu peut arrêter le serveur de messagerie et compter les secondes qui le séparent du premier coup de fil d'un utilisateur furieux.

Architecture de la messagerie

La figure 12.7 décrit l'architecture d'une messagerie Internet classique. À gauche, un utilisateur qui émet un message, son poste de travail, et son serveur de messagerie. À droite, le destinataire du courrier, son poste de travail, et son serveur de messagerie.

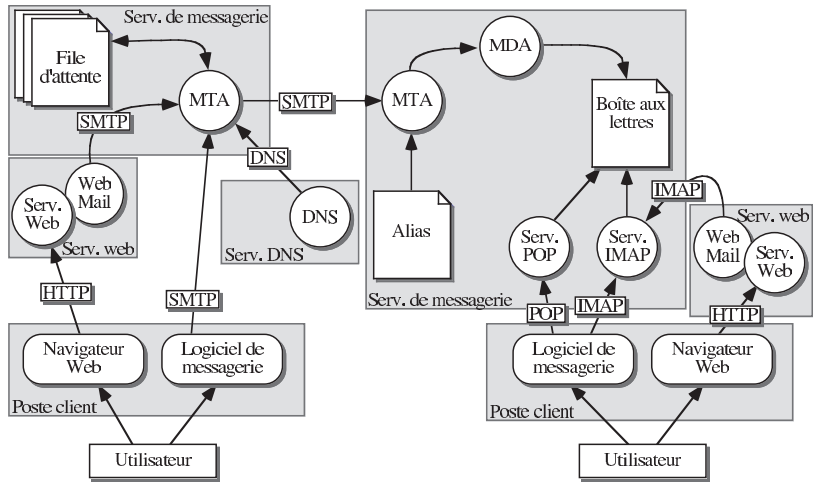


Figure 12–7 Architecture d'une messagerie Internet

Client de messagerie : le MUA

L'utilisateur interagit avec la messagerie par le biais d'un logiciel appelé client de messagerie, ou MUA (*Mail User Agent*). Les plus courants sont Eudora, *Netscape Messenger*, Mozilla *Thunderbird*, et les passoires à virus de Microsoft, baptisées Outlook et Outlook Express.

Serveur de messagerie : le MTA

L'envoi du courrier se fait par le protocole SMTP (*Simple Mail Transfer Protocol*). Le serveur de l'utilisateur émetteur emploie un *daemon* appelé MTA (*Mail Transfer Agent*), qui accepte les messages émis selon ce protocole.

ALTERNATIVE **WebMail**

Le WebMail est un MUA alternatif. Dans ce cas, le MUA fonctionne sur un serveur Web, et seule son interface utilisateur est disponible sur le poste client, à travers un navigateur Web. Vis-à-vis du serveur de messagerie, c'est un MUA comme tout autre.
Dans notre exemple, le serveur Web fonctionne sur une machine distincte du serveur de messagerie, mais ces deux services peuvent très bien être regroupés sur le même système.

ALTERNATIVE **Utilisateurs locaux**

Les utilisateurs travaillant directement dans un shell sur le serveur de messagerie utilisent des MUA qui n'ont pas l'obligation de recourir à SMTP pour dialoguer avec le MTA. Souvent, grâce à une commande d'attribut *set-UID* ou *set-GID*, ils sont capables de placer directement le message à envoyer dans la file d'attente du MTA. Exemples de tels MUA : Mutt, Pine, ELM, ou tout simplement la commande **`mail`**.

ALTERNATIVE **UUCP**

UUCP (*Unix to Unix CoPy*) forme une alternative à SMTP. Antérieur, mais encore utilisé, UUCP est particulièrement apprécié pour transférer le courrier entre sites connectés de façon intermittente à Internet.

Le travail du MTA consiste, comme son nom l'indique, à transférer le courrier vers la machine où se situe le compte de messagerie du destinataire. Ce transfert se fait le plus souvent avec le protocole SMTP, mais pas toujours. Si pour une raison ou une autre il est impossible de transférer le message, le MTA le laisse dans sa file d'attente. Il y reviendra régulièrement.

On trouve de nombreux MTA sous Unix. L'implémentation historique, la plus utilisée, s'appelle Sendmail. Sa complexité – que nous allons bientôt découvrir – et de nombreuses failles de sécurité ont provoqué l'apparition de nombreux autres MTA. Les plus connus sont Postfix, Qmail, et Exim.

Routage du courrier sur Internet avec les MX

Le serveur de messagerie peut contenir des règles de routage du courrier, par exemple pour forcer le passage par un MTA intermédiaire chargé du filtrage anti-virus. Mais comment découvrir à quelle machine transmettre le courrier à destination d'un domaine pour lequel on ne dispose d'aucune règle précise ? Un mécanisme est nécessaire.

Ce mécanisme, c'est le type MX (*Mail eXchanger*) du DNS. Pour savoir à quelle machine envoyer le courrier destiné à un domaine, le MTA de la machine émettrice interroge le DNS avec une requête qu'on peut soi-même tester en shell. Pour écrire à webmaster@freebsd.org, on écrira :

```
$ host -t MX freebsd.org
freebsd.org mail is handled (pri=10) by mx1.freebsd.org
```

On apprend ainsi que la machine recevant le courrier du domaine freebsd.org est mx1.freebsd.org. Pour envoyer le message, un MTA s'y connectera donc en SMTP.

PIÈGE À C... **Mise à jour du MX**

Si vous devez mettre en place ou déplacer un serveur de messagerie, n'oubliez pas de mettre à jour le MX, ou votre nouveau serveur ne recevra pas le courrier adressé au domaine.

Par ailleurs, ne perdez pas de vue le fait que la modification d'un champ MX dans le DNS n'est pas instantanément visible par tous les MTA susceptibles d'envoyer du courrier. Les jeux de *cache* des serveurs DNS peuvent causer des délais de plusieurs heures dans la propagation d'une modification.

Le plus simple pour gérer ces temps de propagation lors d'une migration de serveur de messagerie, c'est de conserver les deux serveurs en état de marche, et de configurer l'ancien serveur pour tout faire suivre au nouveau.

Si cela n'est pas possible, il existe une autre solution : quelques jours avant la migration, vous pouvez diminuer la durée de vie (*Time To Live*, ou TTL) du champ MX à quelques minutes (rappel : dans le fichier de zones, on peut indiquer un TTL en secondes avant le nom de classe IN). Ainsi, la modification ne prendra pas longtemps à se propager, aucun serveur DNS ne gardant la valeur en cache plus de quelques minutes. N'oubliez pas de rétablir un TTL raisonnable une fois la migration terminée.

DORMEZ TRANQUILLE MX **de secours**

Le MX d'un domaine peut renvoyer plusieurs noms de machines, disposant chacune d'une valeur de préférence. Un MTA essaie toujours de se connecter au MX de préférence la plus basse. En cas d'échec, il tentera les MX de préférences plus élevées, qui sont donc des MX de secours. Ces derniers se contentent souvent de garder le courrier en file d'attente et d'essayer régulièrement de l'envoyer au MX principal.

Si aucun MX n'est disponible, le MTA émetteur conserve le message en file d'attente et fera une nouvelle tentative plus tard. Au bout d'un certain temps (souvent, 5 jours d'échecs répétés), il déclare forfait et renvoie le message à l'expéditeur.

Un MX de secours permet d'éviter de rejeter du courrier suite à une panne de plusieurs jours. On le place de préférence sur un autre site, souvent dans une autre institution (le fournisseur d'accès Internet peut assurer ce service. Les universités s'échangent souvent entre elles le service du MX secondaire).

En cas d'incident réseau vous isolant de l'Internet plusieurs jours, un MX secondaire sur un autre site, toujours connecté, pourra réceptionner le courrier. Il suffit de mettre en place une connexion provisoire et de secours, telle qu'un lien par modem RTC, et on peut contacter le MX secondaire pour récupérer le courrier du site. UUCP est très utile dans cette situation : il serait très difficile d'obtenir le routage vers le réseau par un lien RTC provisoire, le MX secondaire ne pourra donc pas contacter le MX primaire en SMTP. UUCP est conçu pour gérer les liens intermittents, et il permettra au MX primaire de contacter le MX secondaire pour rapatrier le courrier du site.

ALTERNATIVE **Routage de messages**

Plusieurs moyens permettent de router le courrier dans un domaine : les tables d'alias, les tables de routage par nom de domaine, ou en s'appuyant sur les `MX` du DNS.

EN PRATIQUE **Les boîtes aux lettres**

L'emplacement des boîtes aux lettres varie légèrement selon les systèmes Unix. Sur les BSD, on les trouve dans le très classique /var/mail.
Le répertoire /var/mail est en mode 1777. le 1 correspond au drapeau `t`, dit aussi *sticky bit* (bit collant) ; la page **man** `chmod(2)` détaille cette notion.

CULTURE **Les origines de Sendmail**

Sendmail est aujourd'hui produit par le consortium Sendmail, mais à l'origine c'est encore un logiciel qui fut développé à l'Université de Californie à Berkeley (UCB), comme BIND, BSD, et PostgreSQL.

SÉCURITÉ **Autres MTA**

Sendmail a un passé peu glorieux en matière de sécurité. Vieux programme, obligé de fonctionner sous l'identifiant de root, il est destiné à être accessible depuis Internet. Pour couronner le tout, SMTP ne demande aucune authentification, ce qui facilite la tâche aux assaillants.
D'autres MTA ont un passé sans tache côté sécurité : ils ont été écrits récemment, en s'inspirant des problèmes de sécurité de Sendmail. Mais l'absence de trous de sécurité connus peut signifier que le code source est très bien écrit ou bien que le programme n'a pas encore vraiment été éprouvé. Sendmail étant très largement majoritaire, il constitue aujourd'hui une cible plus attractive pour l'indélicat à la recherche d'une faille de sécurité. À mesure que les MTA alternatifs prendront de l'importance, cette situation peut changer.

Distribution locale du courrier : le MDA

Le MTA de la machine réceptrice reçoit le message et utilise éventuellement une table d'alias pour traduire une adresse `prenom.nom@example.net` en `login@example.net`.

Quand il connaît l'identifiant utilisateur de destination, le MTA transfère le message au MDA (*Mail Delivery Agent*), chargé de distribuer le courrier.

Divers MDA ont des fonctions distinctes : dépôt dans un boîte aux lettres ; envoi à une liste, à un programme externe...

Chaque système Unix propose un MDA « standard » pour le dépôt dans les boîtes aux lettres des utilisateurs. Sur les BSD, il s'agit de la commande **`mail.local`**. On peut aussi installer d'autres MDA, comme **`procmail`**, qui permet de trier automatiquement le courrier entre plusieurs boîtes.

Chaque utilisateur qui a reçu du courrier au moins une fois dispose d'une boîte aux lettres à son nom, simple fichier texte regroupant tous les messages qu'il a reçus. Ce fichier est en mode 600 pour éviter que les autres utilisateurs ne puissent le lire. En général, le MDA est capable d'y déposer du courrier car il est *set-UID* root. Toutefois, le MDA n'est que *set-GID* `mail` dans certaines configurations. Les boîtes doivent alors être en mode 660 et appartenir au groupe `mail` pour que le MDA puisse accomplir sa tâche.

Serveurs POP et IMAP

Une fois reçu, le courrier peut être consulté par un MUA fonctionnant sur le serveur de messagerie ; il lui suffira de consulter le fichier de boîte aux lettres.

Les MUA distants utilisent pour rapatrier le courrier les protocoles POP et IMAP, dont nous avons examiné les possibilités au chapitre 9. Il faut bien entendu qu'un serveur POP ou IMAP assure les services correspondants. On en trouve plusieurs sous Unix, les plus répandus étant QPopper et UW-IMAP (qui, comme son nom ne l'indique pas, inclut aussi un serveur POP).

La configuration de Sendmail

L'implémentation originale et la plus répandue de la messagerie sur Internet est Sendmail. C'est aussi un des logiciels les plus difficiles à configurer. Nous allons pourtant aborder cette question, car il est proposé dans la distribution de base des trois BSD et de nombreux autres Unix.

L'organisation de Sendmail

Sendmail s'organise autour de quelques fichiers de configuration et d'une seule et unique commande : **`sendmail`**. Elle peut invoquer le MTA sous forme de *daemon*, examiner l'état de la file d'attente, la vider, ou tester le fichier de configuration. Tout dépend des options qu'on lui fournit. Quelques exemples :

sendmail -bd -q30m	L'option **-bd** invoque un *daemon* **sendmail** qui reçoit les requêtes SMTP du réseau. **-q30m** lui indique de traiter les messages en file d'attente toutes les 30 minutes.
sendmail -q	Traite immédiatement les messages en attente.
sendmail -bp	Affiche le contenu de la file d'attente.

Un fichier de configuration maître, /etc/mail/sendmail.cf, contient les emplacements de tous les autres fichiers utilisés par Sendmail.

Pour ses listes d'alias, de règles de routage, ses listes noires, et bien d'autres choses encore, Sendmail utilise des fichiers binaires, compilés à partir des fichiers texte. Leurs noms sont définis par le sendmail.cf.

Dernier point important avant d'aborder le fichier de configuration de **sendmail** : les journaux. **sendmail** utilise **syslogd**, et envoie tout dans la catégorie `mail`. **syslogd** est en général configuré pour consigner tout cela dans /var/log/maillog.

Un aperçu de l'enfer : le fichier sendmail.cf

Le fichier sendmail.cf est le pire fichier de configuration que vous pourrez manipuler sous Unix. Très volumineux (24 kilooctets sans les commentaires), sa syntaxe est particulièrement opaque. Exemple :

```
SLocal_check_mail
Scheck_mail
R$*                     $: $1 $| $>"Local_check_mail" $1
R$* $| $#$*             $#$2
R$* $| $*               $@ $>"Basic_check_mail" $1
SBasic_check_mail
R$*                     $: < ${deliveryMode} > $1
R< d > $*               $@ deferred
R< $* > $*              $: $2
R$*                     $: $1 $| $>"tls_client" $&{verify} $| MAIL
```

Une rapide visite du fichier devrait rapidement vous convaincre du fait qu'il sera difficile d'en maîtriser la syntaxe. Heureusement, cela ne sera pas nécessaire : peu d'administrateurs écrivent leur sendmail.cf à la main.

Sendmail prévoit toutes les situations possibles et imaginables. Il est ainsi très configurable et très puissant, mais peu d'options de configuration interviennent pour une utilisation normale. On peut donc fabriquer automatiquement un fichier de configuration à partir d'une spécification plus simple de ce que l'on désire obtenir.

ASTUCE **Suivre les journaux**

Il est souvent utile de suivre le fichier maillog en temps réel pour observer le bon ou mauvais fonctionnement de la messagerie. Ceci est possible avec la commande **tail -f /var/log/maillog** : elle fait défiler la fin du fichier à mesure que **syslogd** y ajoute des lignes.

EN PRATIQUE **File d'attente**

Sur les systèmes BSD, la file d'attente de Sendmail se trouve dans /var/spool/mqueue.

SUR LES AUTRES UNIX **Le fichier de configuration de sendmail**

On trouve parfois le fichier de configuration sous /etc/sendmail.cf.
On peut aussi trouver un fichier /etc/sendmail.fc, version compilée en binaire de /etc/sendmail.cf. Il était utilisé sur de très anciennes versions de Sendmail pour améliorer les performances.
Ces versions antédiluviennes de Sendmail sont criblées de trous de sécurité : ne connectez surtout pas un tel système à Internet sans avoir fait une mise à jour de Sendmail au préalable.

AU SECOURS ! **Quelle horreur !**

Ces lignes sont des règles de réécriture, qu'on utilise pour transformer les adresses et prendre les décisions de routage. Rassurez-vous : il n'est pas nécessaire de les comprendre pour administrer Sendmail.

PLUS LOIN **Le fichier d'alias**

Le fichier où l'administrateur intervient le plus souvent est la table d'alias, souvent placée dans /etc/mail/aliases. Extrait de fichier aliases :

```
Emmanuel.Dreyfus:       manu
John.Doe:               jdoe@example.com
```

On le voit, ce fichier permet de traduire une adresse de type `prenom.nom` en un nom d'utilisateur Unix, ou de faire des redirections de messagerie.
À chaque modification du fichier aliases, il faut régénérer sa version binaire aliases.db, seule réellement utilisée par **sendmail**. On emploie pour cela la commande **sendmail -bi**, dont **newaliases** est un alias.

Le kit Jussieu

Le kit Jussieu est produit par Pierre David, Jacky Thibault, et Sébastien Vautherot. Il comprend une excellente documentation sur Sendmail et son fichier de configuration, bien plus complète et précise que la brève introduction de ce chapitre. Il propose aussi un script de génération automatique qui, à partir de fichiers très simples, produira un fichier sendmail.cf abondamment commenté en français.

Le fichier suivant permet par exemple au kit Jussieu de générer un sendmail.cf pour un serveur de messagerie contenant des boîtes aux lettres d'utilisateurs et utilisant un relais SMTP pour envoyer tout le courrier vers l'extérieur :

```
Host='khazad-dum'
Domaine='example.com'
ListeDomaines='example.com'
AdressesLocales='HOST'
AdressesInternes='RIEN'
RelaisExterieur='smtp.[mx.example.com]'
MailhostEnInterne='OUI'
TableRoutages='hash -N /etc/mail/routage'

MailerLocal='/usr/libexec/mail.local DFMPlmnrs mail -d $u'
MailerUucp='/usr/bin/uux DFMhmsu uux - -r $h!rmail ($u)'

Aliases='/etc/mail/aliases'
RevAliases='hash -N /etc/mail/revaliases'
SendmailSt='/var/log/sendmail.st'
SendmailHf='/etc/mail/helpfile'
Mqueue='/var/spool/mqueue'

CodeErreur='551 451'
ReecritureAdressesLocales='example.com'
```

La documentation du kit Jussieu, dont la lecture est recommandée, décrit les différentes variables. Ce kit est disponible à l'adresse suivante : http://www.kit-jussieu.org.

ALTERNATIVE **Macros m4**

L'équipe de Sendmail recommande officiellement de construire le fichier de configuration avec un jeu de macros, compilé par la commande **m4**. La documentation de Sendmail détaille cette méthode.

EN BREF **Configuration de Sendmail**

Bref résumé de cette partie : pour faire fonctionner Sendmail, on créera le fichier sendmail.cf avec le kit Jussieu, et on invoquera le *daemon* **sendmail** comme suit : **sendmail -bd -q30m**. Un Sendmail 8.12 ou plus récent requerra de plus la commande **sendmail -Ac -q30m** mais en principe les scripts de démarrage s'en chargent.

Séparation de privilèges dans Sendmail 8.12 et au-delà

À l'origine, les MUA fonctionnant dans le shell utilisaient la commande **sendmail** pour envoyer le courrier. **sendmail** était *set-UID* root, ce qui lui permettait de déposer le message dans la file d'attente sans avoir été invoqué par un utilisateur privilégié.

Les programmes *set-UID* root ont mauvaise réputation car ils constituent des risques : une erreur de programmation dans un tel programme peut permettre à un utilisateur d'obtenir des privilèges de root.

À partir de la version 8.12, l'équipe de Sendmail a modifié le fonctionnement de l'envoi local des messages et supprimé le besoin d'un programme *set-UID* root.

Dans les versions récentes, l'envoi local de messages par **sendmail** dépend d'un nouveau fichier, submit.cf. **sendmail** n'envoie plus le message dans /var/spool/mqueue, mais au serveur SMTP précisé dans ce fichier (par défaut, `127.0.0.1`).

S'il ne peut contacter ce serveur, **sendmail** déposera le message dans une file d'attente particulière : /var/spool/clientmqueue. Cela ne requiert pas les droits de root, contrairement à /var/spool/mqueue, cette nouvelle file est accessible

en écriture au pseudo-utilisateur `smmsp` (*SendMail Mail Submission Program*, ou programme de soumission du courrier de SendMail). **`sendmail`** n'est plus que *set-GID* `smmsp`, ce qui lui permet d'écrire dans la file d'attente et d'être moins dangereux en cas d'erreur de programmation.

Une instance de **`sendmail`** surveille régulièrement la file d'attente /var/spool/clientmqueue et tente de se connecter au serveur SMTP indiqué dans submit.cf pour écouler les messages. L'ensemble du dispositif est décrit dans la figure 12.8.

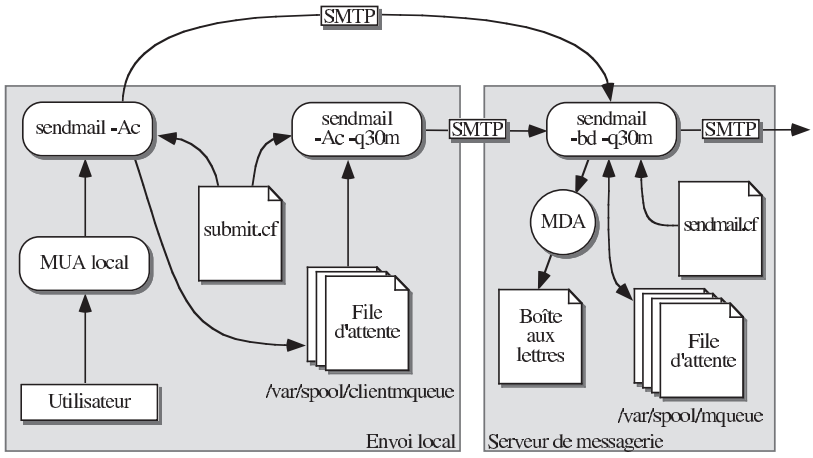


Figure 12–8 Envoi local à partir de Sendmail 8.12

Cette nouvelle disposition a perturbé les habitués de Sendmail : l'envoi de courrier local ne se fait plus aussi simplement qu'avant. Il faut utiliser une instance de **`sendmail`**, qui traite la file d'attente /var/spool/clientmqueue, et submit.cf doit indiquer un serveur SMTP capable d'acheminer le courrier.

Les utilisateurs du kit Jussieu conserveront le submit.cf fourni dans la distribution de Sendmail (en ajustant éventuellement le serveur SMTP à la main) : seul sendmail.cf est généré par le kit.

PLUS LOIN **Choisir le mode d'opération**

En principe, **`sendmail`** devine seul, en fonction du contexte, à partir de quel fichier il doit se configurer. Parfois, il faut toutefois lui expliciter le fichier de configuration pertinent, comme pour vider une file d'attente particulière.
Invoqué avec l'option **`-Ac`**, **`sendmail`** utilise submit.cf. Avec **`-Am`**, il lit sendmail.cf.

ATTENTION **Traitement périodique de /var/spool/clientmqueue**

En principe, les scripts de démarrage de **`sendmail`** s'en chargent, mais en cas de besoin, l'instance traitant la file d'attente /var/spool/clientmqueue s'invoque ainsi : **`sendmail -Ac -q30m`**.
Sous OpenBSD, le traitement du /var/spool/clientmqueue est assuré par l'invocation périodique de la commande **`sendmail -Ac -q`** par la crontab de root.
Un piège à c… notoire : ne jamais invoquer cette instance de **`sendmail`** a pour effet d'oublier en file d'attente les messages envoyés localement alors que le serveur SMTP était indisponible.

Lutte contre les *spams* et les virus

Le *spam*, ou courrier publicitaire non sollicité, est le fléau de la messagerie sur Internet : puisqu'ils ne coûtent presque rien à envoyer, même un très faible taux de retour les rentabilise. Ces messages envahissants peuvent occuper la majeure partie du trafic d'un serveur de messagerie, et la situation semble empirer de jour en jour.

Les virus circulant par courrier électronique présentent un danger : pendant les pics d'infection virale, le serveur de messagerie pourra atteindre les limites. Outre les problèmes de sécurité sur les postes infectés, la messagerie risque la panne (ce qui a le bon goût de stopper momentanément la propagation de l'épidémie, il faut le reconnaître).

CULTURE **L'étymologie du spam**

Le terme *spam* désignait à l'origine une sorte de porc en conserve (*Spiced Pork and hAM*). Les courriers publicitaires indésirables ont été baptisés *spam* en référence à une saynète des Monty Python, où un couple essaie sans succès de commander dans un restaurant un menu sans *spam* – ils en contiennent évidemment tous : « *You can't have egg, bacon, spam and sausage without the spam* ».

Les dispositifs de filtrage se répandent donc sur les serveurs de messagerie : le but est de tenter de détecter une partie des *spams* (sans détruire de véritables courriers), et si possible la totalité des virus.

Bien balayer devant chez soi

Les émetteurs de *spam* (spammeurs) sont bien organisés. Contrairement à que l'état de la messagerie un lundi matin pourrait laisser croire, ils ne sont pas très nombreux, mais disposent d'outils efficaces leur permettant d'indisposer tout Internet.

Une de leurs recettes classiques, c'est de trouver des serveurs de messagerie acceptant de relayer le courrier : ils les utiliseront comme intermédiaires, et pourront ainsi émettre leurs messages depuis des centaines de machines différentes à la fois, plutôt que depuis leurs propres machines. Cela complique la constitution de listes noires.

Le premier pas de toute lutte *anti-spam* consistera donc à s'assurer que le serveur ne se comporte pas en relais ouvert. Le serveur de messagerie du site `example.com` n'a par exemple aucune raison d'accepter un message transitant de `sales@boss.com` vers `manu@netbsd.org`.

Le kit Jussieu, si vous y recourez, vous aura mitonné des règles anti-relais aux petits oignons. Toute autre manière de configurer Sendmail devra prendre en compte cette question.

ASTUCE **Mais comment les spammeurs ont trouvé mon adresse ?**

La première intuition est de penser qu'on a été vendu par son fournisseur de services Internet. C'est rarement le cas. Les spammeurs ont recours à des robots collecteurs d'adresses électroniques, aussi appelés spambots.
Ces derniers écument le Web et Usenet, engrangeant tout ce qui ressemble à une adresse électronique. Une bonne contre-mesure pour éviter le spam consiste par exemple à faire disparaître toute adresse électronique de son site Web. À l'heure où sont écrites ces lignes, on peut conserver un annuaire d'adresses électroniques derrière des formulaires de recherche, car les spambots n'ont pas encore appris à les remplir. On peut aussi remplacer les adresses par des formulaires Web d'envoi de messages.

EN PRATIQUE **Je me fiche de relayer, ce spam ne va pas chez moi !**

Mauvais raisonnement… Tout relais finira par être utilisé pour propager du *spam*, et sera donc inscrit sur les listes noires d'un certain nombre de sites. Tout courrier issu d'un tel serveur, même valable, risque d'être bloqué.

Tester le relais

Une simple session SMTP sous **`telnet`**, comme décrit au chapitre 9, permet de tester si un serveur est un relais ouvert. Exemple d'un serveur rejetant correctement une tentative de relais :

```
$ telnet mail.example.com 25
Trying 192.0.2.15...
Connected to mail.example.com.
Escape character is '^]'.
220-InterScan Version 3.6-Build_1201 $Date: 01/16/2002 22:21:0048$: Ready
220 mail.example.com ESMTP SendmailEAT 8.11.6/8.11.6; Sun, 25 May 2003 16:26:28
+0200;.
HELO mail.example.net
250 mail.example.com Hello mail.example.net [192.0.2.1], pleased to meet you
MAIL FROM: sales@boss.com
250 2.1.0 sales@boss.com... Sender ok
RCPT TO: manu@netbsd.org
554 <manu@netbsd.org>... Relay operation rejected
```

Ce test simple mettra en lumière une erreur grossière, mais il existe hélas plusieurs manières de relayer. On s'assurera donc de la correction d'une configuration avec un testeur automatique, comme celui de http://www.abuse.net/relay.html.

On peut aussi s'inspirer de la sortie du testeur de www.abuse.net pour réaliser ses propre tests. Exemple de test sur un serveur correctement configuré :

```
Mail relay testing

Connecting to mail.example.net for anonymous test ...

<<< 220 mail.example.net ESMTP Postfix (Postfix Rules!)
>>> HELO www.abuse.net
<<< 250 mail.example.net

Relay test 1

>>> RSET
<<< 250 Ok
>>> MAIL FROM:<spamtest@abuse.net>
<<< 250 Ok
>>> RCPT TO:<relaytest@abuse.net>
<<< 554 <relaytest@abuse.net>: Recipient address rejected: Relay access denied

Relay test 2

>>> RSET
<<< 250 Ok
>>> MAIL FROM:<spamtest>
<<< 250 Ok
>>> RCPT TO:<relaytest@abuse.net>
<<< 504 <spamtest>: Sender address rejected: need fully-qualified address

Relay test 3

>>> RSET
<<< 250 Ok
>>> MAIL FROM:<>
<<< 250 Ok
>>> RCPT TO:<relaytest@abuse.net>
<<< 554 <relaytest@abuse.net>: Recipient address rejected: Relay access denied

(...)
```

La sortie complète du test de www.abuse.net est un peu monotone. Voici un résumé rapide des tests qui sont effectués :

Adresse source (`MAIL FROM:`)	Adresse destination (`RCPT TO:`)
<spamtest@abuse.net>	<relaytest@abuse.net>
<spamtest>	<relaytest@abuse.net>
<>	<relaytest@abuse.net>
<spamtest@example.net>	<relaytest@abuse.net>
<spamtest@[192.0.2.15]>	<relaytest@abuse.net>
<spamtest@example.net>	<relaytest%abuse.net@example.net>
<spamtest@example.net>	<relaytest%abuse.net@[192.0.2.15]>
<spamtest@example.net>	<"relaytest@abuse.net">
<spamtest@example.net>	<"relaytest%abuse.net">
<spamtest@example.net>	<relaytest@abuse.net@example.net>
<spamtest@example.net>	<"relaytest@abuse.net"@example.net>
<spamtest@example.net>	<relaytest@abuse.net@[192.0.2.15]>
<spamtest@example.net>	<@example.net:relaytest@abuse.net>
<spamtest@example.net>	<@[192.0.2.15]:relaytest@abuse.net>
<spamtest@example.net>	<abuse.net!relaytest>
<spamtest@example.net>	<abuse.net!realytest@example.net>
<spamtest@example.net>	<abuse.net!realytest@[192.0.2.15]>

PLUS LOIN **Intoxication des spambots**

L'économie du spam repose sur la capacité à toucher beaucoup de destinataires rapidement et pour un faible coût. L'un des piliers de cette économie est la capacité des spammeurs à collecter facilement beaucoup d'adresses valides grâce à leurs spambots. Plus les spambots fourniront de mauvaises adresses, moins les spammeurs seront efficaces.

Vous pouvez donc nuire aux spammeurs en proposant sur votre site Web des pages remplies de fausses adresses électroniques. Les spambots les enregistreront stupidement, abaissant ainsi la qualité de leur travail.

Pensez à déconseiller aux robots honnêtes de consulter ces pages, pour éviter qu'elles ne finissent indexées dans les moteurs de recherche. Pour cela, créez un fichier robots.txt à la racine de votre site Web et indiquez-y les branches interdites aux robots :

```
User-agent: *
Disallow: /fausses-adresses
```

Les robots des moteurs de recherche honoreront cette interdiction, ceux des spammeurs l'ignoreront, et c'est ce qui causera leur perte.

Dispositifs de filtrage

Abordons la question du filtrage des *spams* et des virus. Les solutions abondent ; voyons-en quelques-unes.

Listes noires

C'est le dispositif le plus simple. Le kit Jussieu permet de constituer un fichier d'adresses à rejeter absolument, très utile pour refuser une bonne fois pour toutes les adresses comme big@boss.com, un temps synonymes de virus.

On peut aussi utiliser des bases de données de spammeurs proposées sur Internet. Ces dispositifs ont le gros avantage de répercuter immédiatement tout ajout dans la base sur tous les sites qui y font appel. Le kit Jussieu prévoit également l'utilisation de telles bases.

AMaViS

AMaViS (*A MAil VIrus Scanner*) est un programme qui se charge de décortiquer les pièces jointes des messages qu'on lui transmet, décompactant et décodant tout ce qui peut l'être.

Il fait ensuite appel à un anti-virus pour conclure ou non à une infection. On trouve des anti-virus propriétaires et commerciaux pour les systèmes Unix ; leur liste est donnée sur le site Web du projet AMaViS : http://www.amavis.org. Ils sont distribués sous forme de binaires, et pour la plupart ne proposent pas de version pour les BSD, mais la compatibilité binaire Linux suffit à les faire fonctionner (le chapitre 11 explique comment la mettre en œuvre).

Mentionnons des projets de logiciels libres d'anti-virus : OAV, écrit en Java, et Clam, écrit en C, qui utilise la base de description des virus du premier.

Enfin, il est important de mentionner le coût important du filtrage anti-virus en termes de performances. Le serveur doit en effet décoder et extraire toutes les pièces jointes des messages, et certaines d'entre elles peuvent être de taille importante.

IMPORTANT **AMaViS n'est pas un anti-virus**

Il faut bien comprendre qu'AMaViS n'est pas un anti-virus, mais une interface entre la messagerie et les anti-virus. Non accompagné d'un anti-virus, AMaViS n'a en général aucun intérêt.

EN PRATIQUE **Clam et OAV**

Les sites Web de ces deux projets fourniront plus d'informations : http://clamav.elektrapro.com pour Clam, et http://www.openantivirus.org pour OAV.

PLUS LOIN **Bases de données de spammeurs**

Certaines bases de données sont stockées dans le DNS ; le projet SpamCop en propose une. Si par exemple l'adresse IP `192.0.2.15` a été enregistrée comme appartenant à un spammeur dans la base de SpamCop, le DNS connaîtra l'adresse `15.2.0.192.bl.spamcop.net`.
Pour plus d'information sur SpamCop, consultez le site du projet : http://spamcop.net. SpamCop est une liste très agressive : des fournisseurs d'accès grand public s'y retrouvent régulièrement piégés. Il se peut donc que son service soit excessif par rapport à vos attentes.

On trouve sur Internet des annuaires de listes noires ayant pour vocation la lutte contre le *spam*. Exemple d'un tel annuaire : http://www.declude.com/junkmail/support/ip4r.htm.
Ces bases de données ont leurs limites : les spammeurs pourraient voler momentanément des blocs d'adresses IP avec de fausses annonces de routes en BGP (*Border Gateway Protocol*, le protocole des routeurs des systèmes autonomes, comme expliqué au chapitre 10). Peu de temps après le forfait, les adresses apparemment émettrices d'un *spam* seront redevenues celles d'un malheureux qui n'y est pour rien, et qui se retrouvera piégé dans les listes noires.

SpamAssassin

SpamAssassin est un filtre *anti-spam*. Il utilise de nombreuses règles pour déterminer si un message semble ou non être un *spam*. Ces règles produisent un « score de *spam* », il suffit ensuite de décider d'un niveau de seuil à partir duquel les messages seront marqués comme *spam*, ou même éventuellement détruits. Il est évidemment possible de réajuster la pondération des tests.

Les algorithmes employés par SpamAssassin ne sont que des heuristiques : ils marqueront parfois comme *spam* des courriers anodins (faux positifs) et réciproquement (faux négatifs). Il convient donc de rester vigilant, et d'éviter les suppressions silencieuses de messages sans vérification par un être humain. La pratique la plus courante consiste à mettre les messages marqués comme étant potentiellement du *spam* dans un dossier spécial, et de faire le tri régulièrement.

Dilemme : ce qui est du *spam* pour l'un peut être un courrier désirable pour l'autre. SpamAssassin propose un mécanisme de filtrage bayésien, où l'utilisateur indique a posteriori au filtre ce qui finalement était du *spam* et ce qui n'en était pas. C'est assez contraignant dans la mesure où il faut renvoyer chaque message au filtre pour son apprentissage, mais cela permet d'obtenir un filtrage sur mesure.

Pour finir sur SpamAssassin, il faut mentionner le coût en termes de performances : il est élevé, les filtres devant parcourir plusieurs fois l'ensemble du message. Le fait que SpamAssassin ne soit pas écrit dans un langage compilé n'arrange guère les choses.

Pour plus d'informations, voyez le site Web de SpamAssassin : http://www.spamassassin.org

Filtrage par mots clefs

Il s'agit d'une solution de filtrage très simple, mais qui peut rendre de fiers services. Un certain nombre de programmes permettent de traiter un courrier en fonction de sa conformité à des expressions régulières. **`procmail`** et **`milter-regex`** peuvent assurer ces fonctions pour Sendmail. Dans d'autres MTA, comme par exemple Postfix ou Exim, cette fonctionnalité est intégrée, et aucun programme externe n'est nécessaire.

`procmail` peut être utilisé pour classer le courrier sur plusieurs boîtes, voire pour faire traiter certains messages par des scripts externes. On peut aussi tout simplement l'utiliser pour envoyer vers /dev/null les messages indésirables. Voici par exemple un fichier de configuration .procmailrc qui détruit les messages contenant une ligne caractéristique d'un virus :

```
:0 B
*Dear friend , use this Internet Explorer patch now!
/dev/null
```

PIÈGE À C... **Fausses alertes**

Bien entendu, les filtres à expressions régulières feront disparaître tout message citant une expression qui doit être interceptée. En règle générale, on préfère émettre une notification lorsqu'un message est ainsi détruit. Ceci peut se faire avec **`procmail`**, moyennant de renvoyer le message vers un script externe. Reportez-vous à la page **`man`** de **`procmailrc`** pour découvrir comment faire.

milter-regex se limite à accepter ou rejeter les messages, avec notification à l'expéditeur. Exemple de configuration du fichier milter-regex.conf pour éliminer le virus Swen.A :

```
reject Message contains the Swen.A virus
body /ZGUuDQ0KJAAAAAAAAAB\+i6hSOurGATrqxgE66sYBQfbKATvqxgG59sgBLerGAdL1zAEA6sYBWPXV/
```

Pour bien comprendre d'où sort cette suite de caractères improbables, il faut regarder à quoi ressemble un message électronique contenant une pièce jointe binaire. Voici un extrait d'un virus :

```
(...)
--rqELHosAenypUSf
Content-Type: application/x-msdownload; name="q417088.exe"
Content-Transfer-Encoding: base64
Content-Disposition: attachment

TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAuAAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5vdCBiZSBydW4gaW4gRE9TIG1v
ZGUuDQ0KJAAAAAAAAAC3Egfb83NpiPNzaYjzc2mIGmxkiPJzaYhSaWNo83NpiAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAFBFAABMAQMAdV1OPgAAAAAAAAAA4AAPAQsBBgAAgAAAAPABAAAAAACEEQAA
ABAAAACQAAAAAEAAABAAAAAQAAAEAAAACQAfAAQAAAAAAAAAAIACAAAQAABa7QIAAgAAAAAAEAAA
EAAAAAAQAAAQAAAAAAAAAAAEAAAAAAAAAAAAAAtI0AACgAAAAAsAAAbMABAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAoAgAAIAAA
AAAQAACcAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAALnRleHQAAAAYfwAAABAAAACAAAAAEAAA
AAAAAAAAAAAAAAAAIAAAYC5kYXRhAAAA6BIAAACQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEAAAMAu
cnNyYwAAAGzAAQAAsAAAANABAACQAAAAAAAAAAAAAAAAABAAABABXyhORAAAAAAAAAAAAAAAAE1T
(...)
```

Les pièces jointes binaires sont encodées dans des formats leur permettant de traverser tous les serveurs de messagerie sans encombre, y compris ceux qui ne prévoient que le transit de messages en format texte. Les formats les plus classiques sont base64, UUcode, et Macbinary. Tous trois transforment un document binaire en document composé uniquement de caractères affichables.

Moyennant de filtrer une ligne du virus encodé, on peut l'éliminer à moindres frais, le serveur de messagerie ne devant même pas décoder le message. S'il s'agit d'un virus très actif, que l'on reçoit plusieurs centaines de fois par minute, cette économie devient primordiale.

Dans le même souci d'économie des ressources, on procédera à ce genre de filtrage avant d'effectuer des opérations plus coûteuses, comme le traitement par SpamAssassin ou AMaViS.

ASTUCE Filtrage des exécutables Windows

Une méthode radicale pour éliminer les virus, consiste tout simplement à refuser les exécutables Windows. Ceci peut se faire en se basant sur l'extension du fichier, mais la liste des extensions de fichiers exécutables semble interminable. Une alternative consiste tout simplement à filtrer sur l'en-tête des exécutables Win32 et Win16 encodée en base64 :

- `/^TVqQAAMAAAAEAAAA/`
- `/^TVoAAAEAAAACAAAA/`
- `/^TVpQAAIAAAAEAA8A/`
- `/^TVqTAAEAAAACAAAQ/`

CULTURE Le virus Swen.A

Swen.A est passé relativement inaperçu du grand public, mais il restera gravé dans les mémoires de certains habitués de Usenet comme étant le logiciel diabolique de l'année 2003.

Les usagers d'UNIX se rient habituellement des virus, dont les nuisances sont essentiellement réservées aux utilisateurs de Windows. Avec Swen.A, pas besoin de se faire infecter pour subir les nuisances. Le virus, qui n'infecte effectivement que les postes Windows, se connecte aux serveurs NNTP pour trouver les adresses électroniques de ses nouvelles victimes, auxquelles il s'envoie ensuite par courrier électronique. Jusqu'ici rien d'inquiétant pour celui qui n'utilise pas l'environnement Microsoft. Mais ce virus fait 140 kilooctets et il se réplique à une vitesse effroyable. En pleine infection, l'auteur recevait jusqu'à 1400 copies de Swen.A par jour, de quoi faire sauter n'importe quelle boîte aux lettres normalement constituée, voire tout le serveur de messagerie. Certains fournisseurs de services Internet ont d'ailleurs subi des perturbations très importantes dans l'exploitation de leur messagerie, faute d'avoir filtré ce virus.

Listes grises

Les listes grises (*grey listings* en anglais) sont une version temporaire des listes noires, pour lutter contre le spam et les virus. Chaque message se présentant au MTA est consigné dans la liste grise, puis refusé avec une erreur temporaire. Le comportement normal du MTA émetteur du message sera de tenter une nouvelle transmission quelque temps après. À la deuxième transmission, le MTA retrouvera trace du message dans sa liste grise, et l'acceptera si un temps configuré par l'administrateur s'est écoulé depuis la première tentative.

Lors du traitement des messages normaux, les listes grises introduisent simplement un délai supplémentaire à l'acheminement du message. Pour les spams et les virus, la situation est différente : les moteurs SMTP qui les envoient n'essayent pas en général de retransmettre les messages ainsi refusés, mais se contentent d'essayer une autre adresse.

Bien sûr, il est inévitable que les spammeurs s'adaptent à ce dispositif. Mais entre les tentatives que nous lui imposons, le spammeur ne restera pas inactif, et tentera d'envoyer d'autres messages. En lui imposant un délai, nous augmentons les chances de le voir envoyer un message à une adresse pot de miel d'une liste noire distribuée avant sa deuxième tentative. Si c'est le cas, à cette dernière, l'émetteur sera identifié comme spammeur grâce aux listes noires, et son message sera rejeté.

`relaydelay` est un filtre à listes grises pour Sendmail, qui nécessite l'installation d'une base mySQL. L'auteur du présent ouvrage a mis au point une alternative n'ayant pas besoin de base de stockage externe : **`milter-greylist`**, bien entendu disponible dans les systèmes de paquetages de NetBSD et de FreeBSD.

Pour plus d'informations sur les listes grises, reportez-vous à l'article d'Evan Harris, qui a inventé cette méthode : http://projects.puremagic.com/greylisting/

EN PRATIQUE **`milter-greylist`**

Pour tout savoir sur les dernières évolutions de **`milter-greylist`**, visitez la page du projet : http://hcpnet.free.fr/milter-greylist/

Joe's j-chkmail

Joe's j-chkmail lutte aussi sur le front du *spam* comme sur celui des virus. Son approche est assez différente de celles d'AMaViS et de SpamAssassin.

Contre les virus, il n'essaie pas de filtrer les seuls virus avérés. Tous les virus *potentiels* sont éliminés : *j-chkmail* filtre les exécutables Windows. Cela n'évite pas

PLUS LOIN **Adresses pots de miel**

Les pots de miel (ou *honeypots*) attirent en temps normal les mouches. En informatique, ils attirent les cyberdélinquants.
Le principe d'une adresse pot de miel est simple : il s'agit d'une adresse électronique que l'on publie sur le Web ou sur Usenet, en précisant qu'elle est invalide et que personne ne doit y écrire. Les spambots ignoreront évidemment cette mise en garde et la collecteront dans leurs bases d'adresses.
On a donc l'assurance que tout courrier reçu à une adresse pot de miel est un spam. Il ne reste plus qu'à transmettre tout message ainsi obtenu à un script qui en inscrira l'expéditeur dans une liste noire. Ceci peut se faire via le fichier aliases de Sendmail, avec une ligne comme celle-ci :

```
Fausse.Adresse:          "|/usr/local/bin/script.sh"
```

Le filtre s'exécutera avec les privilèges du serveur de messagerie, c'est-à-dire le plus souvent ceux du pseudo-utilisateur `daemon`. On peut aussi exécuter le script avec les privilèges d'un utilisateur moins privilégié, en plaçant dans son répertoire personnel un fichier .forward contenant la ligne suivante :

```
"|/usr/local/bin/script.sh"
```

les infections par les macro-virus des fichiers Word et Excel, mais cela élimine au moins tout le reste. Nul besoin de maintenir un anti-virus, et les performances sont nettement supérieures à celles d'un véritable anti-virus, puisque *j-chkmail* n'examine même pas le contenu des fichiers.

j-chkmail détecte le *spam* en se fondant sur les comportements et non le contenu des messages. Une machine tentant d'envoyer trop de messages en un laps de temps trop court sera momentanément placée en liste noire. Si c'est celle d'un spammeur, on diminue ainsi sa capacité de nuisance. Dans le cas contraire, elle essaiera à nouveau plus tard.

Vous trouverez plus d'informations sur *Joe's j-chkmail* sur la page Web du projet : http://j-chkmail.ensmp.fr.

PLUS LOIN **Limitation des connexions**

Les systèmes de limitation des connexions ont pu paraître prometteurs, mais les spammeurs s'adaptent hélas trop vite. La nouvelle génération de moteurs d'envoi de *spam* procède par attaques distribuées. Les messages venant de plusieurs machines différentes, il est beaucoup plus difficile de les bloquer en limitant les connexions.

Validation des adresses d'expédition

Les spammeurs utilisent pratiquement toujours de fausses adresses d'expédition, couvrant ainsi leurs traces. L'encombrement des files d'attente des messageries par des messages d'erreur à destination d'adresses impossibles à atteindre est le corollaire direct de cette pratique.

Pour éviter ces désagréments et bloquer un peu de spam, une solution consiste à vérifier l'adresse de l'expéditeur. Le MTA Exim a cette fonctionnalité intégrée ; **`milter-sender`** la fournit pour Sendmail.

Le principe est simple : à chaque message se présentant au serveur, on commence à tenter d'envoyer un message à l'expéditeur. Les commandes SMTP suivantes sont passées :

```
HELO serveur
MAIL FROM: <>
RCPT TO: <foobarbuz@example.com>
QUIT
```

PLUS LOIN **Adresse <>**

L'adresse source <> est utilisée pour envoyer des notifications de statut de distribution (DSN : *Delivery Status Notification*). Il s'agit de messages indiquant qu'un courrier électronique n'a pas pu être délivré. On utilise cette adresse source pour éviter tout emballement, avec des DSN renvoyées suite à d'autres DSN : sans adresse source, on ne peut pas répondre.
C'est également ce mécanisme qui empêche deux serveurs de messagerie utilisant **`milter-sender`** de se vérifier mutuellement les adresses d'expédition en boucle.

Si au cours de cette transaction une erreur temporaire est renvoyée, le message est refusé avec une erreur temporaire. En cas d'erreur permanente, il l'est avec une erreur permanente. Mais si cette tentative d'envoi ne lève pas d'erreur, le message est accepté.

Cette méthode n'est pas infaillible. Face à un serveur qui accepte apparemment tous les messages pour les accepter ou les refuser ensuite, ce test de DSN donnera toujours un résultat positif. Cette technique permet toutefois d'éliminer une partie des spams. Un programme comme **`milter-sender`** permet d'aller éventuellement plus loin, en envoyant un message à l'expéditeur pour lui demander confirmation de son envoi. Une fois la confirmation envoyée, l'adresse de l'expéditeur est validée, et **`milter-sender`** acceptera les messages qui en émanent pendant un temps configurable. Le programme ASK (*Active Spam Killer* : l'exterminateur de spam – quel programme !) fournit une fonctionnalité similaire.

Pour plus d'informations sur **`milter-sender`**, consultez la page Web de ce projet : http://www.snert.com/Software/milter-sender/.

Validation des dates

Une des sales manies des spammeurs consiste à envoyer des messages datés légèrement dans le futur, pour qu'ils apparaissent tout en haut des boîtes aux lettres (en tout cas, celles qui classent les messages par dates). Un filtrage par date permet d'éliminer les spams qui se laissent aller à cette indélicatesse. Il consiste à refuser tout message portant une date d'émission dans l'avenir.

Pour Sendmail, **`milter-date`** remplit cette tâche.

Que faire du courrier indésirable ?

Aucun dispositif de filtrage n'est fiable à 100 %. Que faire des messages supposés indésirables ?

On évitera de les supprimer sans autre forme de procès : cela peut nuire à la fiabilité de la messagerie. En général, on préfère émettre un message d'erreur. Mais, dans le cas de la lutte anti-virale, ce message d'erreur peut rapidement devenir nuisible : les virus étant très souvent envoyés avec des adresses d'expédition usurpées, les messages d'erreur ne servent plus qu'à polluer les boîtes de malheureux qui n'ont rien à voir avec l'envoi du virus. Nous verrons dans la prochaine section qu'en filtrant au bon endroit, ce problème peut trouver sa solution.

Mise en œuvre des filtres

Comment mettre en œuvre tous ces logiciels ? Sendmail propose deux interfaces simples : l'interception du MDA et Milter.

Interception du MDA

C'est la méthode la plus simple à mettre en œuvre, et elle est peu intrusive : elle consiste à remplacer le MDA par le filtre. Quand celui-ci se sera acquitté de sa tâche, il pourra appeler le MDA original pour l'opération de dépôt dans la boîte aux lettres.

On changera de MDA en modifiant sendmail.cf. Il y est indiqué dans une ligne commençant par `Mlocal`. Exemple :

```
Mlocal,         P=/usr/libexec/mail.local, F=lsDFMAw5:/|@qPrmn9,
                S=EnvFromL/HdrFromL, R=EnvToL/HdrToL,
                T=DNS/RFC822/X-Unix,
                A=mail -d $u
```

Il suffit donc de remplacer /usr/libexec/mail.local par le filtre auquel on veut confier les courriers. Bien entendu, le filtre doit lui-même être configuré pour savoir comment déposer les messages.

Cette méthode a quelques inconvénients :

- Il est difficile d'utiliser plusieurs filtres différents. On peut configurer le premier filtre pour en appeler un second et construire ainsi une chaîne de filtres, mais ce n'est pas toujours facile.

RAPPEL **Le MDA ?**

Le MDA (*Mail Delivery Agent*) est le programme chargé de déposer les messages dans la boîte aux lettres du destinataire. Sur un BSD, c'est normalement **`mail.local`**.

SUR LES AUTRES MTA **Changer le MDA**

On peut aussi remplacer le MDA sur les MTA autres que Sendmail : il suffit de changer sa définition dans les fichiers de configuration.

PLUS LOIN **Mise au point**

Cette méthode permet d'expérimenter le filtre sur un compte de test avant d'en déployer la configuration pour tous les utilisateurs. Durant cette phase de tests, il ne sera pas nécessaire de modifier le fichier sendmail.cf ; il suffira de créer dans le répertoire de l'utilisateur un fichier .forward contenant une ligne telle que :

```
"|/usr/local/bin/filtre"
```

Tout le courrier à destination de ce compte sera renvoyé au programme /usr/local/bin/filtre, qui lui-même pourra filtrer avant d'éventuellement invoquer le véritable MDA pour déposer le message dans la boîte aux lettres.

EN PRATIQUE **Quelle version de Sendmail ?**

Milter a été introduit dans Sendmail 8.11, mais il est recommandé d'utiliser une version 8.12 minimum : Milter n'était pas parfaitement au point dans la version 8.11.

- Certains filtres, conçus spécialement pour Milter, ne peuvent pas être utilisés par cette méthode. C'est par exemple le cas de *Joe's j-chkmail* ou des différents **`milter-date`**, **`milter-regex`**, ou **`milter-sender`**, qui comme leurs noms l'indiquent, sont spécifiques à Milter. En revanche, AMaViS et SpamAssassin ne poseront pas de problème.
- On ne peut filtrer que les messages destinés aux utilisateurs locaux, mais pas les courriers en transit ne passant pas par une boîte aux lettres.
- Parvenu au MDA, un courrier a été officiellement accepté par le serveur de messagerie. Il est trop tard pour rejeter le message (il est toujours possible d'émettre un message d'erreur, mais si l'adresse d'expédition est usurpée, il n'aura pas la destination voulue).

L'interface Milter

Milter, contraction de *Mail filter* (filtre de courrier), est une interface de programmation spécifique à Sendmail qui permet de filtrer le courrier lors de son traitement par le MTA. Ceci permet de filtrer tout message de passage, et même de refuser des transits. Milter peut provoquer un refus définitif ou temporaire, voire, à partir de Sendmail 8.13, provoquer une mise en quarantaine du message.

Le grand point fort de Milter sur l'interception du MDA, c'est la capacité de rejeter le message alors que le MTA n'en a pas encore pris la responsabilité. Ainsi, c'est à l'émetteur d'éventuellement s'occuper d'envoyer un message d'erreur.

Cette nuance peut paraître anodine pour des messages normaux, mais elle prend tout son sens pour des spams et des virus dont l'adresse d'expédition est fausse : on peut ainsi éviter d'envoyer un message d'erreur à l'internaute dont l'adresse a été usurpée et qui n'a rien à voir avec l'envoi du message. Si ce refus est fait au niveau du MTA, sur le MX du domaine, on est directement en prise avec le virus ou le moteur d'envoi de spam, et c'est à lui que l'on notifiera le refus de prendre en charge son message.

Milter est assez simple à mettre en œuvre, à condition que la version de Sendmail le prenne en charge. La mise à jour de Sendmail étant très intrusive, Milter pourra se révéler plus délicat à mettre en œuvre que la substitution du MDA.

PIÈGE À C... **Notification aux adresses usurpées**

Depuis le printemps 2004, l'Internet est entré dans une nouvelle ère, celle de la pandémie virale. Les raisons sont multiples : monoculture microsoftienne rendant la quasi-totalité des machines vulnérables à la même attaque, généralisation des accès à haut débit pour les particuliers, qui permettent aux virus de mieux se diffuser, et manque d'éducation chronique des nouveaux arrivants sur Internet, incapables de se protéger efficacement.
À l'heure où cette deuxième édition part à l'imprimerie, les nouvelles versions des virus Netsky et Beagle continuent à apparaître sur une base parfois journalière, et bien sûr, les adresses des expéditeurs sont toujours usurpées. Renvoyer des alertes au virus à l'adresse d'expéditeur employée par le virus est donc devenu une pratique à proscrire, puisqu'elle aboutira dans 100% des cas à une mauvaise adresse.
Si vous tenez vraiment à notifier quelqu'un et que vous ne pouvez pas filtrer sur le MX au niveau du MTA, limitez-vous au destinataire.

Un filtre Milter se présente sous la forme d'un *daemon* qui communique avec **sendmail** via une *socket* UNIX ou un port TCP (on peut donc le placer sur une autre machine que **sendmail** lui-même). On indique sa présence à **sendmail** en ajoutant quelques lignes à la fin du fichier sendmail.cf. Exemple pour un double filtrage par AMaViS et SpamAssassin :

```
O InputMailFilters=amavis, spamass
Xamavis, S=local:/var/amavis/amavis.sock, F=T
Xspamass, S=local:/var/spamass/spamass.sock, F=T
```

Les fichiers /var/amavis/amavis.sock et /var/spamass/spamass.sock sont les *sockets* UNIX permettant de communiquer avec les Milter.

Parmi les rares inconvénients de Milter, on peut citer sa spécificité à Sendmail (à moins que ce ne soit une faiblesse des concurrents de Sendmail ?), et sa limitation aux filtres qui ont été conçus pour lui.

Certains filtres, comme *Joe's j-chkmail*, **milter-greylist**, **milter-date**, **milter-regex**, ou **milter-sender** sont conçus en tant que Milter. D'autres, comme SpamAssassin ou AMaViS, utilisent un *daemon* d'interfaçage avec Sendmail.

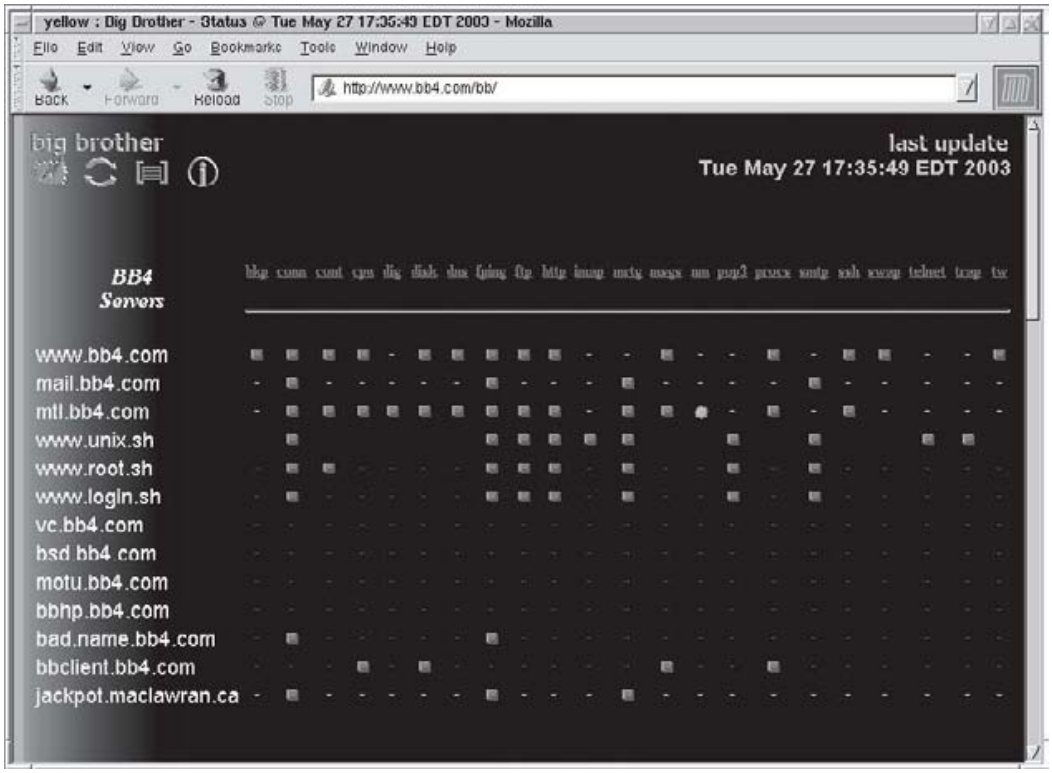
PIÈGE À C… **Décès des daemons**

Les filtres Milter sont très pratiques, mais ajoutent chacun un *daemon* à la messagerie : en leur absence, le courrier cesse d'être distribué.
La configuration permet de préciser si l'absence d'un *daemon* Milter doit provoquer une erreur temporaire (le courrier reste en file d'attente sur la machine émettrice, et sera renvoyé plus tard), ou permanente (le courrier est retourné à l'envoyeur avec une erreur). Dans l'exemple de cette section, **F=T** sélectionne l'erreur temporaire. **F=R** spécifie une erreur permanente, et l'absence d'option **F=** indique que l'erreur doit être ignorée : tout se passe comme si le filtre n'avait pas été configuré.

PLUS LOIN **Socket UNIX**

Les *sockets* UNIX sont un moyen de communication inter-processus limités à la machine locale. Pour communiquer, les processus utilisent un fichier spécial : un processus y écrit, et l'autre y lit. La *socket* se comporte comme un tuyau entre les deux processus. On les appelle d'ailleurs souvent tubes nommés (*named pipes*).

13


yellow : Big Brother - Status @ Tue May 27 17:35:49 EDT 2003 - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://www.bb4.com/bb/
big brother
last update
Tue May 27 17:35:49 EDT 2003
BB4 Servers
www.bb4.com
mail.bb4.com
mtl.bb4.com
www.unix.sh
www.root.sh
www.login.sh
vc.bb4.com
bsd.bb4.com
motu.bb4.com
bbhp.bb4.com
bad.name.bb4.com
bbclient.bb4.com
jackpot.maclawran.ca

L'étude des catastrophes

Traiter des problèmes de disponibilité et de sécurité dans un chapitre dédié est difficile : ce sont des domaines variés et transversaux, qu'il faudrait aborder dans tous les chapitres – nous nous y sommes efforcés. Certains points sont forcément passés au travers des mailles du filet, et seront évoqués ici.

SOMMAIRE

MOTS-CLEFS

- Piratage
- Sauvegardes et restaurations
- Surveillance
- Mises à jour

Connaissez-vous vos adversaires ?

Vous connaissez sans doute vos objectifs en matière de sécurité et de fiabilité : une informatique disponible, garantissant la confidentialité des données privées (boîtes aux lettres des utilisateurs, la plupart de leurs fichiers, bases de données de clientèle, système de paie...). Les exemples de données dont la confidentialité, la disponibilité et l'intégrité sont à assurer ne devraient pas manquer dans votre environnement professionnel.

En revanche, connaissez-vous vos adversaires, ceux qui vont tenter de mettre en échec la sécurité de vos systèmes, en en volant les données sensibles, en les modifiant subtilement, ou en les détruisant ? Voici une liste hélas non exhaustive de vos ennemis et de leurs intentions.

Les *crackers*

CULTURE **Cracker ou hacker ?**

En anglais, on parle de *crackers* et de *hackers*, avec de subtiles nuances pas forcément connues de tous : le *cracker* est un pirate informatique pénétrant illégalement dans les systèmes ; le *hacker* est un informaticien doué bricolant ses programmes ou disséquant ceux des autres. Certains *hackers* s'intéressent aux façons de briser la sécurité des systèmes informatiques, mais uniquement sur leurs machines, dans le but de corriger les problèmes et d'améliorer ainsi leur sécurité.
On parle parfois des *crackers* comme des chapeaux noirs (dans les westerns, ce sont les méchants), et des *hackers* comme des chapeaux blancs (les gentils).

Il y a différents types de pirates, le *cracker* étant au bas de l'échelle. C'est typiquement un mineur irresponsable et pas très compétent techniquement, qui s'introduit dans les systèmes grâce à des scripts écrits par d'autres pour exploiter des failles connues. Il s'appelle Jean-Kevin, il habite en région parisienne, et un site Web entier lui est dédié : http://www.zipiz.com.

Ses objectifs sont divers : le frisson de l'interdit, frimer devant les copains, constater avec satisfaction que l'informatique n'est pas si solide que le prétend la publicité, ou tout simplement utiliser vos ressources informatiques et financières. Pour atteindre ces objectifs, il pourra entrer dans le système, voire plus si affinités : détruire les données, utiliser les serveurs pour stocker des fichiers illicites, téléphoner aux frais de votre employeur, voler des numéros de cartes bancaires, ou tout simplement utiliser votre réseau comme camp de base pour mener de nouvelles attaques.

Le plus souvent, il n'innove pas, mais se contente d'utiliser des scripts déjà faits – on parle de *script-kiddie* (gamin aux scripts). Pour l'empêcher d'entrer, il suffit donc de boucher les trous de sécurité au fur et à mesure de leur découverte et publication. Nous verrons comment procéder plus loin.

Il est plus dur de se protéger d'un pirate innovant, puisqu'il va exploiter une faille inconnue qu'il a découverte seul. Dans ces situations, on peut espérer détecter l'intrusion à défaut de l'empêcher : la surveillance du système prend alors toute son importance.

Les anciens employés et autres ennemis intimes

Plus dur à contrer que le *cracker* : l'ancien employé qui veut se venger d'avoir été licencié. Il connaît bien le système informatique et peut même disposer de mots de passe d'administration. On le contrera en tenant à jour les comptes d'utilisateurs : un tel personnage ne devrait plus avoir accès à aucune ressource informatique.

Parfois, les choses se corsent : l'ennemi intime fait encore partie de l'entreprise. Ne supposez jamais qu'une faille de sécurité inaccessible de l'extérieur ne sera

pas exploitée de l'intérieur : c'est la source de la majorité des attaques ! Il convient donc d'adopter les mêmes règles de sécurité pour toutes les machines du réseau.

L'espion industriel

Pire encore : l'espion industriel. Il a les compétences et les moyens, en veut à vos informations, et s'est juré de les obtenir. Bien plus innovant qu'un *cracker*, il aura au moins la politesse de ne pas semer le désordre dans le système informatique : en restant discret, il pourra revenir.

À défaut de s'en protéger, on peut espérer déceler son activité en surveillant le système informatique. Ce profil d'attaquant est l'illustration du fait que les intrusions ne sont pas toujours accompagnées de perturbations fracassantes.

Les vers, virus, et chevaux de Troie

Même si les auteurs des vers, virus et chevaux de Troie ne ciblaient pas spécialement vos systèmes, leurs créations n'en pourront pas moins les perturber. Une machine compromise par un ver ou un cheval de Troie n'est plus digne de confiance : elle peut héberger un *rootkit*. Il faudra la réinstaller complètement, en ayant bien pris soin au préalable d'identifier la faille qui a causé l'intrusion, sous peine de devoir recommencer rapidement.

Les mesures prophylactiques contre les vers sont les mêmes que celles qui ciblent les *crackers* : corriger les trous de sécurité connus. On évitera les chevaux de Troie en contrôlant les sommes de contrôle ou signatures cryptographiques des exécutables et scripts d'origine douteuse que l'on doit invoquer sur la machine.

Les virus posent surtout problème sur plateforme Windows, et concerneront plutôt vos utilisateurs. Ils se cachent dans les secteurs de démarrage des disquettes, dans les macros des documents Word, Excel et PowerPoint, et plus récemment dans les pièces jointes exécutables envoyées par courrier électronique. Ils font perdre un temps considérable en maintenance informatique : il faut désinfecter les postes sinistrés.

La reproduction des vers et virus peut consommer une partie significative des ressources informatiques, au point d'empêcher tout travail. Mesure d'urgence : déconnecter les machines infectées tant qu'elles ne sont pas soignées. Mesure de prévention : tenir des antivirus à jour sur les postes. Sur un grand parc, cela peut s'avérer difficile : on en placera alors sur les serveurs de fichiers, mandataires (*proxies*) Web, et serveurs de messagerie.

Vos propres utilisateurs

Vos utilisateurs représenteront souvent une faille de sécurité importante, à leur insu. Certains inconscients téléchargent force jeux idiots ou utilitaires inutiles sur Internet et s'échangent quantité d'images, de musiques, de vidéos. Les risques

B.A.-BA **Vers, virus, et chevaux de Troie**

Le ver est un programme qui se propage de système en système en exploitant des failles de sécurité dans les services qu'ils offrent. Une fois le système pénétré, il permet souvent à ses auteurs de prendre le contrôle du système.

Le virus n'a pas besoin de failles, c'est l'utilisateur qui l'aidera à s'introduire dans un système informatique, d'où il pourra se propager ailleurs. Cette reproduction est d'ailleurs souvent sa seule motivation.

Le cheval de Troie requiert lui aussi l'aide involontaire de l'utilisateur pour s'introduire dans le système informatique. Sa motivation rejoint celle du ver : il vise à ouvrir l'accès du système à ses auteurs.

CULTURE **Les rootkits**

Les vers, les chevaux de Troie, et les pirates installent souvent des *rootkits*. Il s'agit d'un ensemble de commandes truquées qui remplacent les commandes d'administration de la machine : **`ps`**, **`netstat`**, **`ls`**, etc. Leur but est de camoufler l'activité des pirates : **`ps`** ne montrera pas leurs processus, **`netstat`** ignorera leurs connexions réseau, et **`ls`** n'affichera pas leurs fichiers.

Des outils tels que Tripwire ou AIDE (disponibles dans les paquetages) utilisent des sommes de contrôle pour détecter les altérations les plus grossières des exécutables des commandes. Mais ils ont leurs limites : eux aussi peuvent être modifiés par un attaquant…

De plus, en cas d'activité suspecte, remplacer une commande par une version saine ne suffit pas forcément : le noyau du système lui-même peut avoir été modifié. Une réinstallation complète de la machine (à partir de fichiers dignes de confiance) s'impose donc.

ATTENTION **Risques juridiques**

Les violations des droits d'auteur sont de loin les délits les plus courants sur un système informatique.
L'administrateur n'est heureusement pas responsable des actes de ses utilisateurs, mais il peut tenter de les informer convenablement. La charte d'utilisation des moyens informatiques, signée par chaque nouvel utilisateur, est souvent l'occasion de rappeler les règles juridiques encadrant l'usage du système informatique.

CULTURE **La loi de Murphy**

Les anglophones ont personnalisé avec humour ce que les physiciens appellent « loi de l'emmerdement maximum » ; ils l'appellent « loi de Murphy », et les sites Web multiplient les collections d'exemples humoristiques. On peut la résumer en « si quelque chose est susceptible de mal tourner, alors cela tournera nécessairement mal ».
Elle a un fond de vérité mais souvent une origine psychologique : après tout, qui remarque que la file d'attente d'à côté avance moins vite que la sienne ?

ASTUCE **Choisir un bon mot de passe**

Dans certains environnements professionnels, les pots (de départ, d'accueil, ...) sont monnaie courante. L'alcool et l'informatique ne faisant pas bon ménage (même si certains prétendent avec humour qu'on ne peut pas bien administrer un système Unix entièrement à jeun), assurez vos arrières : choisissez un mot de passe assez complexe pour ne pas être en mesure de le taper lorsque vous ne serez plus en état d'exercer vos fonctions.

sont multiples : exécution de chevaux de Troie, saturation des moyens informatiques professionnels à des fins ludiques, et pour couronner le tout, risque juridique, les contenus échangés l'étant parfois en violation des droits d'auteur.

Vous veillerez donc aux quotas par utilisateur sur les serveurs de fichiers et dans les boîtes aux lettres, à limiter la taille des courriers électroniques transitant dans la messagerie, voire la bande passante attribuée aux indélicats qui empêchent les autres de travailler.

Des éditeurs de logiciel

Tout cela ne suffit pas : vous affronterez encore des logiciels programmés en dépit du bon sens, dans un mépris total de la sécurité. Le pire, c'est que ces programmes sont souvent plébiscités par les utilisateurs, qui refuseront l'idée même d'utiliser autre chose. L'auteur accueillera avec plaisir toute solution à ce problème.

Le destin

Ajoutons une pierre à l'édifice : le destin lui-même, qui se montre parfois facétieux. Il fera par exemple mourir un disque dur du serveur de messagerie une heure avant votre départ en vacances. Seule solution : la redondance. Mais le destin est insistant, et n'hésite pas à provoquer plusieurs pannes en même temps.

Vous-même ?

Enfin, la clef de voûte de ce monument consacré à vos ennemis : vous-même. C'est vous qui allez exécuter un **`rm -Rf`** malheureux à la racine, oublier la rotation d'un journal, provoquant la saturation prématurée du serveur de messagerie un jour crucial, vous tromper dans la configuration des services et des filtres, ouvrant une voie royale aux *crackers* même les plus incompétents.

L'absence d'intention de nuire ne fait pas vraiment de vous un ennemi, mais retenez qu'en tant qu'être humain vous êtes faillible : la sécurité dépendant fortement de vous, elle ne pourra jamais être parfaite. Le risque zéro n'existe pas, mais rien n'empêche de le réduire au maximum.

Quels dispositifs de sécurité pour quels enjeux ?

Quelles armes sont à la disposition de l'administrateur dans cette lutte de Sisyphe ? Le glaive et le bouclier valent ici aussi : il n'y a pas de protection absolue, les attaques évoluent sans cesse. Heureusement, l'arsenal est assez varié et vous permettra de vous battre sur plusieurs plans contre les catastrophes : récupération, détection, et prévention.

Récupérer les catastrophes : sauvegardes

La destruction du système est parfois inévitable : vous avez oublié de colmater une faille, et Jean-Kevin en a profité pour exécuter « **`delete from clients;`** » dans la base de données, espérant que ce fait divers ferait la une. Ou alors le destin farceur a durement frappé un disque dur. Trêve de lamentations, il faut reconstruire. Oui, mais comment ? En utilisant vos sauvegardes, bien sûr. Si vous n'en avez pas, c'est bien fait : il aurait fallu y penser plus tôt. Vous y penserez dans votre nouvel emploi.

CULTURE **SQL**

SQL (*Standard Query Language*, ou langage de requêtes standardisé) est le principal langage de manipulation des bases de données. Ici, "**`delete from clients;`**" ordonne d'effacer tout le contenu de la table **`clients`**.

Sur quoi sauvegarder ?

Il vous faut des sauvegardes. Plusieurs solutions : la plus simple consiste à recopier quotidiennement les données sur un second disque dur, avec un fichier `crontab`. Cela protégera contre la mort d'un disque dur, et c'est sans doute la méthode qui permet la récupération la plus rapide. Mais elle ne protège en rien contre l'attaque d'un pirate, qui détruira sans état d'âme les données du disque de sauvegarde.

C'est pourquoi on a recours aux supports extractibles : disquette Zip ou Jaz, cartouches magnéto-optiques, CD-ROM, DVD, etc. Moyennant de changer régulièrement le support, cette méthode permet de mettre les données à l'abri. On peut aussi stocker plusieurs sauvegardes, pour pouvoir revenir en arrière d'une journée, d'une semaine, d'un mois... L'enjeu est de taille si l'on ne découvre que tardivement une altération des données : il sera possible de retrouver les données originales.

Les CD-ROM ou DVD ont malheureusement une capacité (respectivement de 700 Mo ou 6 Go) faible devant celle des disques modernes, qui dépasse parfois la centaine de giga-octets. La qualité des supports extractibles a son importance, et les disques de mauvaise qualité retiendront en particulier l'information moins longtemps. Certains ont vu des données s'évanouir d'un CD-ROM après un simple séjour de 6 mois dans une boîte à la cave.

Les bandes magnétiques offrent une capacité plus importante : une cartouche DAT permet de stocker 24 Go de données, pour moins de 15 euros. Pour les plus grandes capacités, on peut se tourner vers les bandes DLT, ou investir dans un robot capable de changer les bandes.

Quel que soit le moyen de sauvegarde retenu, il est bon de conserver plusieurs sauvegardes, et sur de longues périodes : une sauvegarde par jour pour le dernier mois, une sauvegarde par semaine pour le dernier trimestre, etc. Pensez aussi à placer les sauvegardes dans un autre local voire sur un autre site, pour éviter qu'une catastrophe (incendie, inondation) ne détruise de concert originaux et copies. Vous pourrez chiffrer des sauvegardes et les confier à un administrateur ami par échange de bons procédés, recourir à la salle des coffres d'une banque, etc. L'idée est qu'une catastrophe suffisamment importante pour détruire simultanément toutes les instances de vos données vous posera des problèmes plus urgents que leur récupération.

Les sauvegardes avec dump et restore

On trouve de nombreux outils de sauvegarde. Le principe est toujours le même : créer une archive contenant tous les fichiers, voire seuls les fichiers qui ont changé après une date donnée (sauvegardes dites « incrémentales »).

Les outils disponibles sont nombreux : **tar**, et le couple **dump** et **restore** sont simples. On trouve aussi des logiciels bien plus complets, libres, comme Amanda, ou propriétaires, comme *Legato Networker*.

Pour exploiter les sauvegardes depuis un autre système d'exploitation, **tar** est probablement la seule possibilité, les commandes **dump** et **restore** de certains systèmes utilisant un format particulier.

Exemples d'utilisation de **dump** et **restore** :

```
mt rewind ❶
dump -0uf /dev/nrst0 /var ❷
( ssh khazad-dum /sbin/dump -1uf - /home ) | dd of=/dev/nrst0 ❸

mt asf 1 ❹
restore -tvf /dev/rnst0 ❺
mt asf 1
dd if=/dev/rnst0 obs=512 | restore -xf - ❻
```

❶ Rembobinage de la bande (par défaut, la première disponible : /dev/rst0).

❷ Sauvegarde du /var de la machine locale. L'option **-0** précise une sauvegarde de niveau zéro (complète) ; **-u** demande une mise à jour du fichier /etc/dumpdates, et **-f** précise la bande de travail ; /dev/nrst0 (par opposition à /dev/rst0) évite le rembobinage automatique.

❸ Sauvegarde à distance à travers une connexion SSH, incrémentale, de niveau 1, du /home de la machine `khazad-dum`.

La sauvegarde via SSH peut se faire sans intervention humaine avec des clefs RSA. Sinon, il faudra taper un mot de passe.

❹ Déplacement sur la deuxième sauvegarde de la bande (la numérotation commence à zéro).

❺ L'option **-t** de **restore** permet d'obtenir le contenu d'une sauvegarde. Pour extraire les fichiers, utilisez l'option **-x**.

❻ Astuce : parfois **restore** proteste à la restauration à propos de la taille des blocs de la bande magnétique. **dd** évite cela.

PLUS LOIN **Sauvegardes incrémentales**

La situation la plus confortable est celle où l'on peut sauvegarder toutes les nuits la totalité des disques de toutes les machines. C'est rarement possible, pour des raisons d'espace disponible sur les supports de sauvegarde, et de temps nécessaire à la copie des données.

On recourt donc aux sauvegardes incrémentales. Par exemple, on sauvegardera le lundi tout le disque d'une machine (sauvegarde dite de niveau 0), et se contentera les autres jours de la semaine de traiter les seuls fichiers qui ont changé depuis la dernière sauvegarde complète (sauvegarde dite de niveau 1).

La récupération commence alors par traiter la sauvegarde de niveau 0, puis la dernière sauvegarde de niveau 1. Avantage : on ne consomme que peu de ressources les jours de sauvegarde de niveau 1, et peut donc en profiter pour opérer une sauvegarde de niveau 0 sur une autre machine.

Dans certains cas, on introduit plus de niveaux de sauvegarde : le niveau 2 correspond à la sauvegarde de tout ce qui a changé depuis la dernière sauvegarde de niveau 1, et ainsi de suite. Pour récupérer, on passera la dernière sauvegarde de chaque niveau, en commençant par le niveau 0.

On veillera à tester la procédure de récupération avant la survenue d'une catastrophe.

EN PRATIQUE **Amanda**

Amanda est disponible dans le système de paquetages des BSD, dans la catégorie sysutils pour NetBSD, et misc pour FreeBSD et OpenBSD.

Détecter les catastrophes : surveillance des systèmes

Il est important de pouvoir déceler les catastrophes avant vos utilisateurs ou vos clients. Avec un peu de chance, vous les rattraperez avant que quiconque s'en aperçoive et vous pourrez ainsi donner le change.

Surveiller la disponibilité

Sur le plan de la disponibilité des systèmes, certaines ressources sont à surveiller de près : occupation des processeurs, des disques, utilisation des goulots d'étranglement du réseau. On surveillera aussi la disponibilité des services. Un *daemon* qui plante (rare) ou ne redémarre pas correctement (plus courant, suite à un problème de configuration) peut rendre indisponible un service critique comme la messagerie ou le Web.

Les outils ne manquent pas pour surveiller ces problèmes. On pourra s'appuyer sur la vérification journalière du fichier /etc/daily des systèmes BSD : elle rapporte les occupations des disques et les changements de configuration – très utile pour rattraper au vol une erreur. Il est possible de lui ajouter des vérifications maison si nécessaire.

PLUS LOIN **Redémarrer automatiquement les daemons instables**

On a parfois affaire à un *daemon* qui plante souvent. L'idéal évidemment est de corriger la cause de cette instabilité, mais c'est parfois impossible (manque de ressources pour mener l'analyse, ou logiciel propriétaire). On souhaite alors le redémarrer automatiquement. Sous UNIX System V et GNU/Linux, c'est simple : une entrée dans le fichier /etc/inittab suffit, avec le mot-clef `respawn`. Les BSD ne proposent pas cette fonctionnalité.

Le code suivant démarre en boucle un programme. Il attend dix secondes puis le redémarre à chaque fois que ce programme rend la main. Cette temporisation évite de consommer trop de ressources si le programme en question se termine immédiatement.

Attention, il faut passer en argument le chemin complet du programme à démarrer, et ce dernier ne doit pas passer en tâche de fond.

```
/*
 * Compilation: cc -o respawn respawn.c
 * Usage: respawn /usr/local/bin/daemon -options
 *
 */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <err.h>
#include <errno.h>
#include <syslog.h>
#include <strings.h>

#include <sys/wait.h>

int
main(argc, argv, envp)
        int argc;
        char **argv;
        char **envp;
{
        pid_t child;
        int exitcode;

        openlog("respawn", 0, LOG_DAEMON);
        close(0);
        close(1);
        close(2);

        switch (fork()) {
        case -1:
                syslog(LOG_DAEMON|LOG_ERR,
                    "cannot fork: %s", strerror(errno));
                return -1;
                break;

        case 0:
                break;

        default:
                return 0;
        }
        setsid();

        argc--;
        argv++;

        while (1) {
                child = fork();
                switch (child) {
                case -1:
                        syslog(LOG_DAEMON|LOG_ERR,
                            "cannot fork: %s",
                            strerror(errno));
                        break;

                case 0:
                        (void)execve(argv[0], argv, envp);
                        syslog(LOG_DAEMON|LOG_ERR,
                            "execve %s failed: %s",
                            argv[0], strerror(errno));
                        return -1;
                        break;

                default:
                        syslog(LOG_DAEMON|LOG_NOTICE,
                            "waiting for pid %d\n", child);
                        (void)waitpid(child, &exitcode, 0);
                        syslog(LOG_DAEMON|LOG_NOTICE,
                            "restarting %s in 10 seconds\n",
                            argv[0]);
                        break;
                }
                sleep(10);
        }
        /* NOTREACHED */
        return 0;
}
```

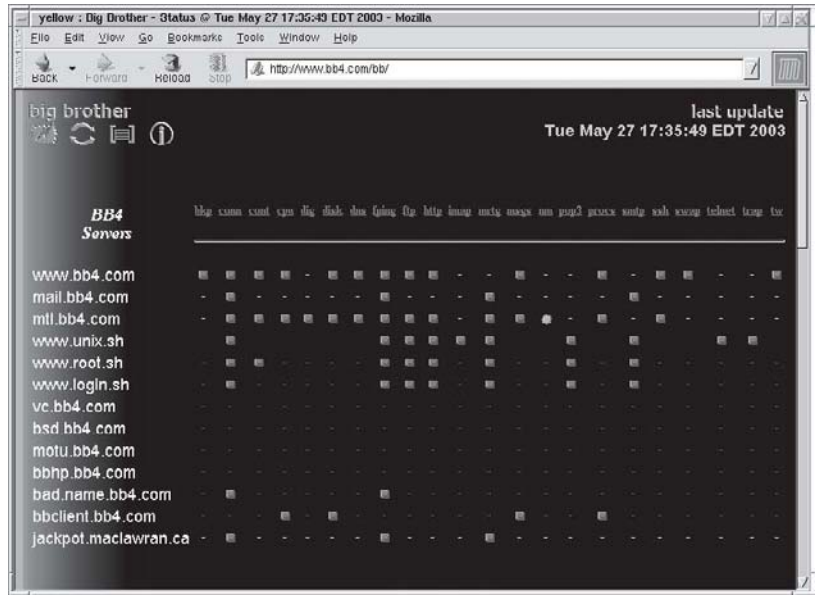


Figure 13–1 Big Brother en action

On peut aussi mettre au point des outils de vérification temps réel, tentant en boucle d'utiliser un service ou de vérifier l'état d'une ressource, et se signalant en cas de problème. Ainsi, l'auteur surveille l'occupation disque de son serveur de messagerie comme suit :

```
$ while [ 1 ] ; do df -k ; sleep 60 ; done
```

Il suffit d'invoquer cette commande dans un **xterm** et de surveiller régulièrement ce qu'elle affiche. Simple, mais efficace. Pour plus complexe, on a le choix entre écrire des petits bricolages à base de scripts shell et d'outils comme **ping** ou **netcat**, ou adopter de gros bricolages déjà tout faits tels que *Big Brother* (figure 13.1). C'est un dispositif de surveillance avec interface Web et possibilité de remonter les alertes par courrier électronique ou SMS.

CULTURE **Le couteau suisse netcat**

netcat (souvent installé sous le nom **nc**) est le véritable couteau suisse du réseau. Il permet d'initier des communications en TCP ou en UDP, en tant que serveur ou en tant que client. Il est très utilisé pour mettre au point des petits scripts de tests.
Il est proposé dans les systèmes de paquetages des BSD, catégorie net.

EN PRATIQUE **La charge processeur**

La charge du processeur peut s'obtenir sous Unix avec les commandes **uptime** et **w** :

```
$ uptime
 7:08PM  up 19 days,  9:34, 1 user, load averages: 0.26, 0.18, 0.11
```

Les chiffres intéressants sont en fin de ligne. Ils représentent le nombre moyen de processus en file d'attente pendant la dernière, les 5 dernières, et les 15 dernières minutes. Un nombre inférieur à 1 indique que la machine passe du temps à ne rien faire.

Plus visuel : des graphiques

On peut encore surveiller l'utilisation d'une ligne spécialisée ou tout simplement l'espace disque disponible en traçant des courbes. Là encore, on peut travailler à la main, avec un script Perl utilisant le module GD pour créer des images ensuite placées sur un serveur Web. Ou faire appel à un produit tout fait comme MRTG, en action dans la figure 13.2.

MRTG visait à l'origine à surveiller l'utilisation des liaisons réseau, mais son champ d'action s'est désormais étendu à la surveillance de la quantité de *spam* interceptée par un serveur de messagerie, comme à la charge CPU d'une machine.

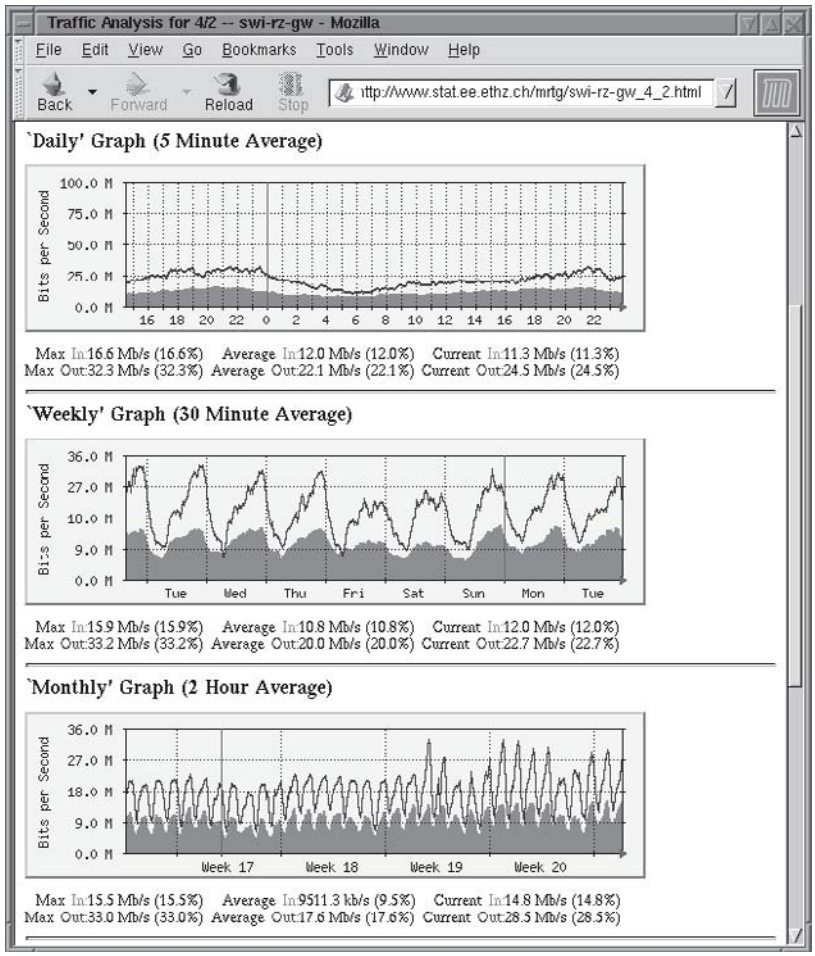


Figure 13–2 MRTG et courbes de trafic

PLUS LOIN **Perl/GD**

GD est une bibliothèque de production d'images, qu'on peut utiliser depuis un script Perl pour créer assez simplement des graphiques. Tous deux sont disponibles dans les systèmes de paquetages des systèmes BSD.

ALTERNATIVE **RRDTools**

Si les capacités d'affichage de MRTG se révèlent insuffisantes pour vos besoins, vous pouvez vous tourner vers d'autres produits. Le logiciel RRDTools permet par exemple de tracer des graphiques beaucoup plus complexes. Il n'assure pas la collecte des données, mais MRTG peut être configuré pour utiliser RRDTools au moment de la génération des graphiques.
MRTG et RRDTools sont disponibles dans les systèmes de paquetages, catégorie net.

Détection d'intrusion : qui rentre chez vous ?

Outils beaucoup plus axés sur la sécurité que sur la disponibilité, les détecteurs d'intrusion sont des programmes qui écoutent le trafic réseau à la recherche d'activités suspectes, comme l'exploration du réseau par un ver ou un pirate.

La façon la plus simple de détecter ce genre d'activités est sans doute de monter une machine qui n'assure aucun service et de la configurer pour journaliser tous les paquets envoyés à l'adresse attachée à son interface réseau. Cette machine, inconnue de tous et silencieuse, ne devrait rien recevoir. Exemple de règle IPFilter qui a cet effet :

```
pass in all
pass out all
pass in log on ex0 from any to 192.168.6.3/32
```

ATTENTION **Récupération des journaux avec IPFilter**
N'oubliez pas de démarrer **`ipmon -s`** et de configurer **`syslogd`** pour consigner les messages **`local0.*`** émis par **`ipmon`** à un endroit où vous pourrez les lire.

Si la machine est directement connectée à Internet (sans pare-feu), vous observerez des dizaines de sondes par jour. Rien d'intéressant, ce ne sont que les *crackers* et les vers qui cherchent une nouvelle victime. Il est beaucoup plus intéressant de surveiller ces activités si elles sont issues du réseau interne : des tentatives de connexion aux ports 22, 25, 80, 139 indiqueront très probablement une activité de recherche de vulnérabilités, ou un collègue qui explore cette nouvelle machine inconnue. Tirez les choses au clair ; si personne ne vous a sondé volontairement, c'est peut-être une machine compromise qui s'est trahie.

On trouve des outils bien plus complets qu'un simple filtre journalisant, tels le détecteur d'intrusion Snort. Gavé de règles décrivant des situations suspectes, il signalera tout ce qui y correspond. L'avantage, c'est qu'on peut l'employer sur une machine assurant un véritable service, voire une passerelle très fréquentée. Snort ne relevant que ce qui selon ses règles est suspect, le bruit du trafic ambiant ne gênera pas.

EN PRATIQUE **Snort**
Snort est disponible dans les systèmes de paquetages des BSD, catégorie net pour NetBSD et OpenBSD, et security pour FreeBSD.

Identifier les causes des catastrophes : les journaux

Les journaux du système (*logs*) sont une source d'information capitale : ils permettent de savoir ce qui s'est passé, et surtout, souvent, qui a fait quoi. Il est donc essentiel de les configurer correctement : conserver les bonnes données, assez longtemps, mais sans prendre trop de place.

Quelles informations garder et pour quoi faire ?

Les données intéressantes sont tout ce qui permet de tracer un abus de la sécurité informatique : connexions et déconnexions des utilisateurs, requêtes sur les serveurs, attributions des adresses IP en adressage dynamique, modifications des adresses MAC associées aux adresses IP, connexions à l'extérieur via un mandataire... Tout ceci pourra permettre d'identifier les causes de certains problèmes, en déterminant qui faisait quoi. En général, les informations sont convenablement

journalisées par les réglages par défaut des *daemons*, mais contrôlez la journalisation de tout nouveau service mis en place, en adaptant la configuration au besoin.

Si par exemple un plaisantin s'amuse dans un courrier électronique à usurper l'identité du président de la société, l'adresse de la machine émettrice apparaîtra dans les journaux, et ses journaux de connexion donneront l'identité des utilisateurs alors connectés. Cela n'est pas une preuve : un compte d'utilisateur suspect peut lui-même avoir été piraté, mais on saura au moins que son mot de passe est compromis ou que la machine est suspecte.

La fiabilité vous intéresse aussi : qu'est-ce qui arrive exactement à un courrier électronique transitant sur les serveurs ? Les utilisateurs accuseront souvent la messagerie de perdre des messages, alors qu'ils n'ont pas lu un message du *Mailer daemon* leur exposant (en anglais) les causes d'un échec. D'où l'importance de pouvoir leur dire que tout a bien fonctionné mais qu'ils n'ont pas tenu compte du message d'erreur de vendredi dernier, 13h42.

Enfin, n'oubliez pas que les utilisateurs ont droit au respect de leur vie privée, même sur le lieu de travail. Les tribunaux le rappelleront durement aux administrateurs manquant de déontologie. La divulgation de certaines informations des journaux ou de ce qui passe dans la messagerie ou sur les serveurs de fichiers de l'entreprise peut valoir une désagréable expérience judiciaire.

Délai de garde des journaux

Il faut les conserver assez longtemps pour pouvoir reprendre de vieilles histoires. Un an est un idéal pas toujours possible, faute de place.

On configurera la rotation des journaux pour éviter que les fichiers ne gonflent au-delà du raisonnable et plombent inutilement les disques et les sauvegardes (un journal d'un giga-octet qui change chaque jour sera en effet intégralement archivé à chaque sauvegarde). Un fichier trop gros risque de plus d'être inexploitable, car trop long à parcourir.

La commande **`newsyslog`**, notamment disponible sur les BSD, a pour tâche de faire tourner les journaux. Au-delà d'une certaine taille ou à un certain âge, au choix, elle change leur nom (transformant par exemple maillog en maillog.1) et peut les compacter. **`newsyslog`** est invoqué par le crontab de root, et on configure son comportement dans /etc/newsyslog.conf. La page **`man`** de ce fichier détaillera tout cela.

La commande **`newsyslog`** détruit les fichiers les plus anciens selon la configuration donnée dans /etc/newsyslog.conf. Attention à ne pas les effacer trop rapidement : il suffit d'attendre qu'ils soient sauvegardés une ou deux fois. Ils pourront ensuite disparaître, puisqu'une copie existera.

DORMEZ TRANQUILLE **Envoyez les journaux ailleurs**

Souvent, les journaux tairont une intrusion : le premier soin d'un pirate est souvent d'effacer ses traces.
On peut s'assurer de leur intégrité en les envoyant sur une autre machine, via le réseau (comme exposé dans les pages **`man`** de syslog.conf et **`syslogd`**). Il est difficile d'altérer les journaux placés sur une machine dédiée, mais on pourra facilement envoyer de fausses informations : **`syslogd`** n'authentifie pas ce qu'il reçoit à distance.
Des programmes comme **`syslog-ng`** ou **`nsyslogd`** évitent ce problème avec des connexions authentifiées et chiffrées par SSL.

CULTURE **Mailer daemon**

Mailer daemon est l'identité utilisée par les MTA (*Mail Transfer Agent*) lorsqu'ils doivent envoyer des courriers électroniques aux utilisateurs pour leur signaler des erreurs dans l'acheminement des messages.

ALTERNATIVE **`logrotate`**

Certains Unix utilisent d'autres outils : **`logrotate`** est un exemple populaire sur les distributions Linux. Sur un Solaris récent, c'est **`logadm`** qui s'acquitte de cette tâche.

PIÈGE À C... **Les journaux qui tournent trop vite**

Par défaut, **`newsyslog`** tourne toutes les heures. Si la vitesse de remplissage du journal est très supérieure à ce qui était prévu initialement (par exemple à cause de messages d'erreur répétitifs), les journaux du matin risquent de disparaître avant même la tombée de la nuit.
On configurera donc **`newsyslog`** pour éviter ce genre de situation. Souvent, le problème se règle tout simplement en éliminant un message d'erreur ou d'avertissement trop souvent répété.

RAPPEL **whois**

Pour retrouver le propriétaire d'une adresse IP ou d'un nom de domaine, on peut consulter les bases de données **whois** de l'ARIN, du RIPE, de l'APNIC, ou du LACNIC. Nous en avons abordé l'utilisation au chapitre 9.

Scénario désagréable

Toutes ces mesures sont utiles, mais un jour, vous vivrez peut-être ceci :

Rien à signaler le matin ; calme plat, *Big Brother* est au vert. Vous jouez à **rogue** (passionnant jeu en mode texte du répertoire /usr/games) quand le téléphone sonne. Un collègue responsable informatique vous explique qu'un plaisantin a remplacé le contenu du site institutionnel de son entreprise par des images d'un goût douteux, et que l'attaque venait d'une de vos adresses IP.

Votre institution n'a qu'un bloc de 8 adresses, utilisées pour les serveurs publics. Toutes ses autres machines sont placées derrière un translateur d'adresses, sur un réseau privé. Évidemment, c'est l'adresse du translateur qui apparaît dans les journaux système du collègue.

Difficile de remonter au coupable. Une piste peut apparaître si vous imposez l'utilisation d'un mandataire pour accéder à l'extérieur. C'est assez contraignant pour les utilisateurs, mais toutes les connexions sur les serveurs Web externes laisseront des traces, dont certaines trahiront peut-être les premiers pas de l'indélicat sur le site distant.

Si les adresses sont attribuées dynamiquement par DHCP pour permettre une mobilité des postes, les journaux du serveur pourront mentionner une nouvelle adresse MAC : on a branché au réseau un ordinateur portable inconnu pour rester anonyme.

Si vous remontez à une machine, ses journaux pourront signaler l'utilisateur Leboulet connecté à l'heure du crime. Las, il est parti en retraite il y a six mois, en vacances au bout du monde, ou chez lui depuis une heure en laissant sa session ouverte. On a espionné son mot de passe alors qu'il le tapait ou profité de sa distraction.

Dans de nombreux cas, vous ne remonterez pas au coupable. Si la société agressée porte plainte, vous participerez à une nouvelle jurisprudence. En France, les fournisseurs d'accès Internet sont désormais considérés comme responsables s'ils ne peuvent pas trouver le coupable d'un délit commis à travers leur réseau. Ils ont l'obligation légale de conserver les journaux un an. Remonter au coupable leur est facile : tout utilisateur est forcé de s'authentifier lors de sa connexion à leur réseau. Pour l'entreprise, la question est plus délicate : il suffit souvent de brancher un portable à la place d'une machine, en lui volant ses paramètres réseau, pour utiliser le réseau de l'entreprise sans s'authentifier. L'administration système est un métier à risques...

Prévenir : filtrages, mises à jour, redondance

Au lieu de détecter et réparer les catastrophes, l'idéal serait de les éviter : mieux vaut prévenir que guérir. Voyons les outils de prévention disponibles.

Redondance des matériels et des services

Mettre en place de la redondance au niveau des disques et des services vous donnera le temps de réparer sereinement une panne alors que le système continuera de fonctionner normalement. Travailler sur un incident informatique sans être continuellement dérangé par les coups de téléphone des usagers impatients est un privilège qu'il faut savoir apprécier à sa juste valeur...

La tolérance aux fautes pour les disques : le RAID

Une panne typique : un disque dur meurt, emportant avec lui bon nombre de données importantes, et stoppant surtout l'activité d'un serveur crucial.

Du côté des données, les dégâts sont limités car vous disposez de sauvegardes. Les données écrites depuis la dernière sauvegarde seront quant à elles perdues. Si le disque hébergeait une file d'attente de messagerie, cette perte de données est un problème.

La panne d'un disque impose également une opération de maintenance précipitée pour le remplacer et restaurer une sauvegarde avant de pouvoir redémarrer le serveur.

Pour se protéger de tels désagréments, on utilise des dispositifs appelés RAID (*Redondant Array of Independent Disks* : matrice redondante de disques indépendants). Leur objectif consiste à répartir en continu l'information sur plusieurs disques, pour pouvoir assurer la continuité du service lors de la panne de l'un d'entre eux.

FreeBSD dispose d'une solution RAID à travers le gestionnaire de volume Vinum. Consultez la page **man** `vinum(4)` pour plus de détails. NetBSD et OpenBSD proposent pour leur part RAIDframe, documenté dans la page **man** `raid(4)`. NetBSD devrait également proposer Vinum à partir de sa version 2.0.

On peut pousser plus loin la logique du RAID en mettant en place des serveurs dédiés au stockage, voire des réseaux de serveurs de stockage. De tels systèmes augmentent la tolérance aux fautes tout en autorisant une mutualisation du dispositif de stockage pour plusieurs machines. C'est là le domaine des NAS (*Network Attached Storage* : stockages accessibles par le réseau) et des SAN (*Storage*

CULTURE **L'étymologie de Vinum**

Vinum est inspiré du système de gestion de volume commercial Veritas. Le proverbe latin « *In vino veritas* » est à l'origine de ce nom – *vino* est le nom *vinum* décliné à l'ablatif en latin.

PLUS LOIN **RAID 0, RAID 1, RAID 5 et autres...**

Il existe de nombreux types de systèmes RAID, certains ajoutant un entrelacement (*stripping* en anglais) de l'information sur plusieurs disques pour assurer des débits plus forts lors des opérations d'entrée-sorties : cela permet la lecture sur un disque pendant le déplacement des têtes de lectures d'un autre disque, qui prépare la prochaine lecture.

Le RAID 0 ne met en place aucune redondance, et se contente d'entrelacer les données. On l'utilisera dans les situations où la tolérance aux fautes est moins prioritaire que les performances.

Le RAID 1 consiste en une duplication totale des informations. Il n'entraîne aucun gain de performances, mais la tolérance aux fautes est assurée simplement.

Le RAID 5 reprend des idées du RAID 0 et du RAID 1 et introduit un enregistrement de la parité pour assurer à la fois une tolérance aux fautes et de meilleures performances. Contrairement au RAID 1, il subit une perte de performances lors de la panne d'un disque.

On trouve aussi des RAID 0+1, 2, 3, 4, 6, 7, 10, et 53. Les curieux pourront se référer aux nombreuses sources de documentation disponibles sur le Web, comme par exemple : http://www.acnc.com/04_01_00.html.

Area Network : réseau de stockages). Les premiers sont des serveurs de stockage fonctionnant sur des réseaux IP avec des protocoles tels que SMB ou NFS, alors que les seconds utilisent un réseau dédié sur fibre optique (liens *Fiber Channel*) et des encapsulations du protocole SCSI. Il existe aussi des systèmes hybrides, utilisant iSCSI (SCSI sur IP).

RAPPEL **SMB et NFS**

NFS (*Network File System*) est le protocole de partage de fichiers natif sous Unix. Son utilisation et sa configuration ont été largement couvertes au chapitre 8.
SMB (*Server Message Block*) est quant à lui le protocole de partage de fichiers et d'imprimantes natif de Windows. Le logiciel Samba permet à une machine Unix de parler le protocole SMB.

Redondance des services

Les RAID permettront d'assurer la continuité du service en cas de panne d'un disque, mais ils ne feront rien en cas de panne d'un serveur.

Pour lutter contre ces incidents, le plus simple est de mettre en place des serveurs redondants, ce qui s'avérera trivial pour certains protocoles. Le DNS est ainsi prévu pour gérer des serveurs redondants. Installer un ou plusieurs DNS secondaires sur un réseau est facile, et rendra un fier service le jour où le DNS primaire tombera. On peut facilement configurer des machines UNIX pour utiliser plusieurs DNS, en retouchant le fichier /etc/resolv.conf

```
nameserver 192.0.2.10
nameserver 192.0.2.50
options timeout:1
```

RAPPEL **MX et courrier électronique**

Le MX (*Mail eXchanger*) est la machine chargée de recevoir le courrier électronique pour un domaine. Son rôle de MX est annoncé à travers le DNS, grâce à une entrée de type MX dans le fichier de zone du domaine. Voyez la section sur le DNS dans le chapitre 12 pour plus de détails.

La messagerie prévoit également une redondance au niveau du MX d'un domaine. On peut installer deux champs MX dans le DNS, chacun ayant la même priorité : si les clients n'arrivent pas à joindre le premier, ils essayeront automatiquement le deuxième. Exemple d'extrait de fichier de zone définissant deux MX redondants :

```
        IN      MX 5    mx2.example.com.
        IN      MX 5    mx1.example.com.
```

ASTUCE **Expiration des entrées rotatives**

Le nombre `1`, indiqué avant le type `IN`, signifie que cette entrée a une durée de vie d'une seconde. Les serveurs DNS faisant du cache ne la retiendront donc pas, et ceci évitera que de gros réseaux utilisent trop longtemps la même adresse.

D'autres services, comme le Web ou le FTP, ne proposent pas de méthodes de redondance. Il est facile de monter des configurations identiques, mais rien n'est prévu dans les protocoles pour que les clients se reportent automatiquement sur un serveur si l'autre est hors service. On peut toutefois monter une redondance à petit prix grâce à un DNS rotatif.

Le DNS rotatif consiste à attribuer plusieurs adresses IP à une même adresse DNS, ce qui se configure assez simplement dans le fichier de zone d'un domaine :

```
www     1 IN    A       192.0.2.30
www     1 IN    A       192.0.2.31
www     1 IN    A       192.0.2.32
```

Le serveur DNS, interrogé sur l'entrée `www`, renverra les trois adresses dans un ordre aléatoire. On obtiendra ainsi une bonne répartition, même pour les clients les moins malins se contentant de la première adresse qui leur est renvoyée.

```
$ nslookup www.example.com
Server:  dns.local
Address:  10.0.2.1

Non-authoritative answer:
Name:    www.example.com
Addresses:  192.0.2.30, 192.0.2.31, 192.0.2.32
```

```
$ nslookup www.example.com
Server:  dns.local
Address:  10.0.2.1

Non-authoritative answer:
Name:    www.example.com
Addresses:  192.0.2.32, 192.0.2.30, 192.0.2.31
```

Cette méthode n'est pas pleinement satisfaisante : si l'un des serveurs est indisponible, certains clients vont tenter de le contacter, échouer, puis éventuellement essayer l'adresse suivante. Les clients les moins évolués pourront abandonner à la première tentative et donc ne pas bénéficier de la redondance.

Pour une redondance transparente pour le client, on peut faire appel à des dispositifs de mandataires, répartissant le trafic entre les différents serveurs. C'est le rôle des dispositifs appelés commutateurs de niveau 7, ou commutateurs applicatifs.

Enfin, pour certains services, il est impossible de mettre en place une redondance facilement. C'est le cas de tous ceux où il y a possibilité d'écriture concurrente, car on ne peut alors se contenter de monter plusieurs serveurs identiques contenant les mêmes données : les modifications faites sur une machine doivent être visibles sur toutes les autres. Le meilleur exemple est sans doute le serveur de bases de données. Dans ces situations, la redondance n'est possible que si l'application sait se répartir sur plusieurs machines. Pour revenir sur les bases de données, cette fonctionnalité est encore balbutiante dans le logiciel libre (avec les produits MySQL ou PostgreSQL). Lorsque la redondance fait partie du cahier des charges, on est donc souvent amené à choisir Oracle.

Redondance des équipements réseau

Dernière catégorie de matériel pouvant tomber en panne : les liens et équipements réseau. Il y a clairement deux cas à distinguer : les pannes sur le réseau local et les pannes sur les connexions vers l'extérieur d'un site.

Les pannes du réseau local restent extraordinairement peu fréquentes. La cause la plus courante est la panne d'électricité, qui prive d'alimentation une armoire de commutateurs. On peut remédier à cela en plaçant des onduleurs pour alimenter les commutateurs, mais l'intérêt reste limité pour un réseau local : à quoi bon faire fonctionner le réseau dont tous les postes sont hors service faute de courant ? L'autre solution, plus onéreuse, consiste à mettre en place deux circuits d'alimentation électrique indépendants et à utiliser des équipements réseau à alimentation redondante.

ALTERNATIVE **Redondance et règles de redirection**

Comme on l'a vu au chapitre 10, une règle de redirection au niveau du pare-feu peut assurer le même rôle qu'un DNS rotatif. Les clients ne voient alors qu'une seule adresse IP pour l'adresse DNS, et c'est le pare-feu qui assure la répartition sur plusieurs machines.

```
rdr fxp0 192.0.2.30/32 port 80 -> 192.0.2.30,
    192.0.2.31,192.0.2.32 port 80 tcp round-robin
```

ASTUCE **Lien de secours**

Une liaison de secours à bas débit ne remplace pas une liaison spécialisée à haut débit, mais peut assurer une connectivité minimum. On pourra par exemple l'utiliser pour échanger le courrier via UUCP avec un MX secondaire, situé sur un autre site, et assurer ainsi le service du courrier électronique malgré la panne.
Si vous laissez d'autres trafics transiter par votre lien de secours, il est important d'éviter son engorgement. Les techniques de limitation de bande passante évoquées au chapitre 10 seront probablement incontournables pour éviter que le trafic Web ne s'impose au détriment des transferts UUCP.

EN PRATIQUE **Daemons invoqués par défaut**

Les nouvelles versions de Sendmail imposent l'utilisation d'une instance de **`sendmail`** pour le traitement de la file d'attente des messages envoyés localement (comme expliqué au chapitre 12). La configuration par défaut comprendra donc peut-être un **`sendmail`**, qui n'assurera aucun service disponible sur le réseau.

Contre les câbles coupés, les commutateurs les plus perfectionnés sont capables de répartir une liaison Ethernet sur plusieurs câbles et de gérer la mise hors service de l'un d'entre eux. Là encore, ce type d'incident reste très rare sur un réseau local.

Enfin, on peut mettre en place une redondance au niveau de la connexion vers l'extérieur. Elle peut prendre la forme d'un lien de secours, comme une liaison RNIS utilisée lorsqu'une liaison spécialisée tombe, ou d'une double connectivité complète. Dans ce cas, vous disposez de deux fournisseurs d'accès à Internet différents, et vous vous comportez en système autonome, avec des routeurs de bordures parlant le protocole BGP, comme cela a été évoqué au chapitre 10.

Prévention des intrusions : sécurisation des systèmes

Point d'excès dans les services

Du côté de la prévention des intrusions, la première règle est de n'employer que les services vraiment nécessaires. Moins le système comptera de services accessibles par le réseau, plus on diminuera les risques d'ouvrir l'accès à la machine à cause d'un service vulnérable ou mal configuré. Ainsi, sur certains systèmes, il faudra fermer de nombreux services inutiles. Sur les BSD, le système est par défaut configuré pour activer un minimum vital de services depuis plusieurs années : **`syslogd`**, **`cron`**, **`inetd`**, et pour OpenBSD, **`sshd`**.

Certains services ne doivent être assurés que localement, tels une file d'attente d'impression ou un serveur X. Il n'est alors pas souhaitable qu'ils écoutent sur le réseau, et tout éventuel trou de sécurité les concernant ne mettra pas le système en péril.

La commande **`netstat -a`** existe sur la plupart des Unix comme sous Windows. Elle indique la liste des ports TCP en écoute (état `LISTEN`) et les ports UDP auxquels sont attachés des processus qui recevront des paquets externes. Il s'agit de limiter cette liste au maximum. Exemple de **`netstat -a`** sur une machine employant quelques serveurs :

```
$ netstat -a
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        State
tcp        0      0  violette.local.ssh     trotsky.local.49157    ESTABLISHED
tcp        0      0  *.ssh                  *.*                    LISTEN
tcp        0      0  *.nfs                  *.*                    LISTEN
tcp        0      0  *.1022                 *.*                    LISTEN
tcp        0      0  *.7100                 *.*                    LISTEN
tcp        0      0  *.sunrpc               *.*                    LISTEN
udp        0      0  *.nfs                  *.*
udp        0      0  *.1021                 *.*
udp        0      0  *.1022                 *.*
udp        0      0  *.sunrpc               *.*
Active UNIX domain sockets
Address  Type   Recv-Q Send-Q Inode     Conn Refs Nextref Addr
 13cb600 stream    0      0 1d32490        0    0        0 /tmp/.font-unix/fs7100
 13cbe40 stream    0      0        0 d009ec40    0        0
 13cbc00 stream    0      0        0 d0099ec0    0        0
 191b0d8 stream    0      0 1727e80        0    0        0 /var/run/printer
 191b018 dgram     0      0        0 d001f440    0 d0059300
 13cb6c0 dgram     0      0        0 d001f440    0 d0088f00
 13cb240 dgram     0      0        0 d001f440    0        0
 13cb180 stream    0      0 198dc98        0    0        0 /var/run/rpcbind.sock
 13cb0c0 dgram     0      0 18e7470        0 d0099780    0 /var/run/log
 13cb000 dgram     0      0 18e73d0        0    0        0 /var/run/syslog
```

Sous BSD, la commande **fstat** permet de trouver le processus attaché à chaque port. Sous GNU/Linux, **fuser** donne la même information, et dans le cas général, **lsof** est capable de donner cette information sur de nombreux systèmes. Tuez le processus assurant chaque service inutile, et empêchez-le de redémarrer automatiquement avec la machine. Le chapitre 5 détaille ce genre d'opération.

Consultez la documentation des services à n'assurer que localement. Par exemple, sous GNU/Linux comme sur les BSD, le serveur X accepte une option **-nolisten tcp** spécifiant de ne pas écouter sur un port TCP. Sous BSD, le serveur **lpd** accepte de même une option **-s** indiquant de ne pas écouter sur le réseau. Sans réseau, ces *daemons* utilisent des *sockets* UNIX pour communiquer, ce qui assure que ces communications restent locales à la machine.

Les services locaux devraient se limiter aux *sockets* UNIX, mais certains programmes en sont incapables. Il est alors souvent possible de les limiter à une adresse IP, qu'on choisira locale : `127.0.0.1`.

Quant aux services locaux qui persistent à écouter sur le réseau, la seule solution est parfois de déployer des règles de filtrage simples. Exemple pour IPFilter :

```
pass in all
pass out all
block in return-rst on ex0 proto tcp from any to 10.0.12.18/32 port 1023
```

Les administrateurs les plus paranoïaques portent ceinture et bretelles : leurs services sont configurés pour n'être accessibles que localement, et des règles de filtrage IP empêchent de les contacter si une erreur de configuration les rendait un jour accessibles.

Zones démilitarisées

Qui ne dispose pas de suffisamment de ressources pour garantir en permanence la sécurité d'un parc bureautique appréciera la possibilité de filtrer à l'entrée de ce réseau. Les possibilités de filtrage et de translation d'adresse sont évoquées au chapitre 10.

Une bonne pratique consiste encore à séparer les serveurs accessibles de l'extérieur du réseau interne, voire les uns des autres. L'objectif : contenir autant que possible toute éventuelle intrusion ou compromission. Les zones réservées aux serveurs s'appellent souvent « zones démilitarisées » (DMZ pour *DeMilitarized Zone*).

On évitera de même, autant que possible, les communications entre serveurs. Une session SSH depuis un serveur compromis risque en effet de fournir les éléments d'authentification sur la machine cible à un éventuel intrus.

Mise à jour des services

Jetons à terre une éventuelle illusion : tous les logiciels proposant une fonctionnalité intéressante contiennent des trous de sécurité. Ils sont en effet écrits par des êtres humains, faillibles. De plus, certains langages de programmation, comme le C, facilitent malheureusement les erreurs.

CULTURE **lpd**

lpd est un serveur d'impression. Il gère les files d'attente locales et peut proposer une file d'impression sur le réseau. On configure **lpd** par le fichier /etc/printcap, comme nous l'avons vu au chapitre 7.

PLUS LOIN **Sockets UNIX et sockets INET**

Les *sockets* permettent les communications entre processus clients et processus serveurs. Les *sockets* INET utilisent IP ; un serveur les employant ouvrira donc un port TCP ou UDP en écoute sur le réseau. Les *sockets* UNIX utilisent des fichiers spéciaux ; un serveur y faisant appel ne sera accessible que depuis la machine locale, avec les droits du fichier spécial.

ATTENTION **Les clients aussi ont des problèmes de sécurité**

Les mises à jour ne sont pas le propre des services disponibles sur le réseau. Tout client qui y cherche des informations est lui aussi potentiellement vulnérable. Pour n'en citer que quelques-uns, un trou de sécurité dans Netscape permettait d'exécuter du code sur la machine du client lorsqu'il affichait une image mal formée : elle pouvait contenir un cheval de Troie. Autre exemple : en 2002, des problèmes ont été découverts dans les fonctions du client DNS de la libc de la plupart des systèmes : un serveur compromis pouvait compromettre à son tour des clients DNS.
Les clients atteints mettent en danger le compte Unix qui les a invoqués. On trouve aussi des trous de sécurité dans les programmes à attribut *set-UID* (voir chapitre 6), qui permettent de passer root même si l'on ne disposait initialement que d'un compte d'utilisateur normal.

CULTURE **libc**

La libc est une bibliothèque contenant la plupart des fonctions de base disponibles en langage C. Elle implémente aussi de nombreuses fonctions utilitaires et les points d'accès aux appels système.

SÉCURITÉ **Scénario noir**

Le scénario de la vague de compromissions dues à une faille découverte par les pirates ne relève pas de la science-fiction. Fin 2003, la FSF et les projets Debian et Gentoo ont eu des serveurs piratés à cause d'un trou de sécurité dans le noyau Linux.

Lors de sa première distribution, un logiciel ne contient aucun trou de sécurité connu. La recherche de failles commence alors, en général par des gens bien intentionnés : toute découverte est publiée, l'équipe de développement applique un correctif et sort une nouvelle version, et les gens mal intentionnés créent un script pour exploiter automatiquement la faille. Des vers se propagent alors grâce au trou de sécurité et des machines non encore corrigées sont automatiquement compromises.

Il s'agit donc, pour éviter toute intrusion, de faire la mise à jour entre l'annonce de la vulnérabilité et l'apparition des programmes qui l'exploitent.

Il arrive parfois qu'une faille soit d'abord découverte par les chapeaux noirs. C'est alors la panique : une vague de piratages déferle, difficile à juguler. Les programmes exploitant les failles dépendant beaucoup du système d'exploitation et du processeur, une plateforme un peu exotique pourra protéger quelque peu dans une telle situation : seuls les systèmes les plus répandus seront d'abord ciblés. Attention : rien n'empêche, dans le cas général, une personne motivée d'adapter l'attaque à tout système qui en vaudra la peine à ses yeux.

Si l'on néglige la relative protection accordée par la rareté du système, la seule issue est de désactiver momentanément le service posant problème, jusqu'à l'arrivée d'un correctif.

Ce qu'il faut retenir, c'est que l'administration d'une machine reliée au réseau impose de tenir le système à jour. Faute de quoi, il est certain que le réseau fera tôt ou tard les frais d'une intrusion, quelle que soit sa configuration. Utiliser un système qui prétend n'avoir eu qu'une seule vulnérabilité à distance dans l'installation par défaut, durant plus d'un certain nombre d'années, ne sauvera personne.

Les sources d'information sur les failles

De nombreux sites recensent les failles de sécurité dans les logiciels, à commencer par le site Web de chaque équipe de développement. Chacun des projets BSD annonce les alertes de sécurité le concernant sur le Web. Ces annonces sont plus ou moins visibles : sur la page d'accueil, rubrique « alertes de sécurité » pour FreeBSD, sur la page d'accueil, rubrique « actualités » pour NetBSD, et dans la page « sécurité » pour OpenBSD. Des projets comme Apache ou Samba annoncent de même leurs failles sur leur site Web.

On trouve aussi des sites plus généralistes, comme celui du CERT : http://www.cert.org, qui se contente des alertes concernant un risque avéré. D'autres sont moins sélectifs, et donc bien plus prolixes ; c'est le cas par exemple de SecurityFocus : http://online.securityfocus.com/bid, qui recense tous les problèmes de sécurité potentiels. Des sites comme le CERT australien (http://www.auscert.org.au) se situent entre ces deux approches.

Il existe encore des listes de diffusion, telles BugTraq, souvent lieux d'annonce des découvertes en matière de trous de sécurité. À vous de trouver la source d'information la plus adaptée à votre cas. Sur les systèmes BSD, le plus simple

est de passer régulièrement sur le site Web. Chaque BSD propose également une liste de diffusion dédiée à la sécurité.

Comment mettre à jour rapidement ?

On a traité des mises à jour du système de paquetages au chapitre 11. Dans le cas des logiciels installés depuis les sources, il suffit d'en télécharger les nouvelles sources, de recompiler, d'installer les nouveaux binaires par-dessus les anciens, et de redémarrer le service le cas échéant.

Plus délicates sont les mises à jour du système d'exploitation lui-même, sujet suffisamment vaste pour lui dédier la deuxième partie du présent chapitre.

RAPPEL **Audit des paquetages**

Sous NetBSD, le paquetage audit-packages procure un audit automatique et quotidien des paquetages installés, et rapporte les problèmes de sécurité du système. Utilisez-le.

Mise à jour du système

Nous allons maintenant aborder les problèmes de mise à jour du système d'exploitation, ainsi que la personnalisation du noyau.

Quoi mettre à jour ?

Lisez soigneusement les détails de toute nouvelle alerte de sécurité, et assurez-vous d'être concerné par le problème. Si par exemple celui-ci met en jeu un *daemon* utilisé par Kerberos, logiciel que vous n'utilisez pas, il suffira de ne pas démarrer ce *daemon* – les prudents pourront même effacer le programme incriminé.

De même, certaines failles concernent des programmes *set-UID* root pas vraiment critiques pour le fonctionnement du système : il suffira alors de leur ôter cet attribut pour recouvrer la sérénité. Dans le cas d'un programme indispensable, cette suppression peut n'être qu'une mesure d'urgence en attendant un correctif.

Si le problème vous concerne, vous devez mettre à jour. Deux voies s'offrent à vous : une mise à jour binaire (pas toujours disponible), ou par les sources. Ces dernières permettent de contrôler le programme employé, et éviteront toute incompatibilité binaire ou de version de bibliothèque. Nous commencerons donc par traiter de la mise à jour par les sources, et réserverons la dernière section du chapitre aux mises à jour binaires.

RAPPEL **Manipulation du drapeau set-UID**

On manipule le bit *set-UID* avec la commande **chmod** :

```
# ls -l /usr/bin/su
-r-sr-xr-x  1 root  wheel  892732 Jan 10 16:13 /usr/bin/su
# chmod u-s /usr/bin/su
# ls -l /usr/bin/su
-r-xr-xr-x  1 root  wheel  892732 Jan 10 16:13 /usr/bin/su
```

L'arbre de sources

Pour recompiler tout ou une partie du système, il en faut les sources. On les trouve sur le site FTP du projet, sous la forme d'un fichier .tgz, qu'on décompacte en général sous /usr/src. L'arbre de sources de NetBSD a l'allure suivante :

```
$ ls -F /usr/src/
BUILDING        bin/            doc/            lib/            share/
CVS/            build.sh*       etc/            libexec/        sys/
Makefile        crypto/         games/          regress/        tools/
Makefile.inc    dist/           gnu/            rescue/         usr.bin/
UPDATING        distrib/        include/        sbin/           usr.sbin/
```

FreeBSD et OpenBSD ont des arborescences de sources similaires. Les alertes de sécurité indiquent la marche à suivre pour les mises à jour : mettre les sources à jour pour inclure le correctif, recompiler, et installer les binaires ainsi générés. En général, on peut se contenter de ne recompiler que l'élément concerné par le problème (programme ou bibliothèque).

ATTENTION **Corrections dans les bibliothèques**

Les corrections de sécurité de certaines bibliothèques impliquent de corriger tous les programmes qui y sont liés statiquement. Un problème de sécurité dans la libc obligera par exemple à recompiler cette dernière ainsi que tous les programmes des répertoires /bin et /sbin liés statiquement.

CULTURE **Branches de développement**

Les équipes maintiennent souvent plusieurs versions en parallèle des logiciels : une branche « de développement » où les programmeurs ajoutent des fonctionnalités (chez les BSD elle s'appelle *-current* ou *CURRENT*), et une ou plusieurs branches « stables », où ils se contentent de corriger les bogues – l'objectif étant de ne pas en ajouter d'autres. Régulièrement, la branche de développement bifurque et crée une branche où les ajouts de fonctionnalités sont gelés : on obtient une nouvelle branche stable après une période de test.
FreeBSD introduit un niveau de branche de plus depuis la version 4.3, en faisant bifurquer des branches *RELEASE* de la branche *STABLE* pour chaque nouvelle version. OpenBSD a une approche similaire, sauf que ces branches sont issues de la branche *-current* : il n'y a pas de branche stable.
Lors du téléchargement de sources des projets BSD, assurez-vous de choisir la bonne branche.
La figure 13.3 donne un aperçu des branches des projets BSD entre 1999 et 2003.

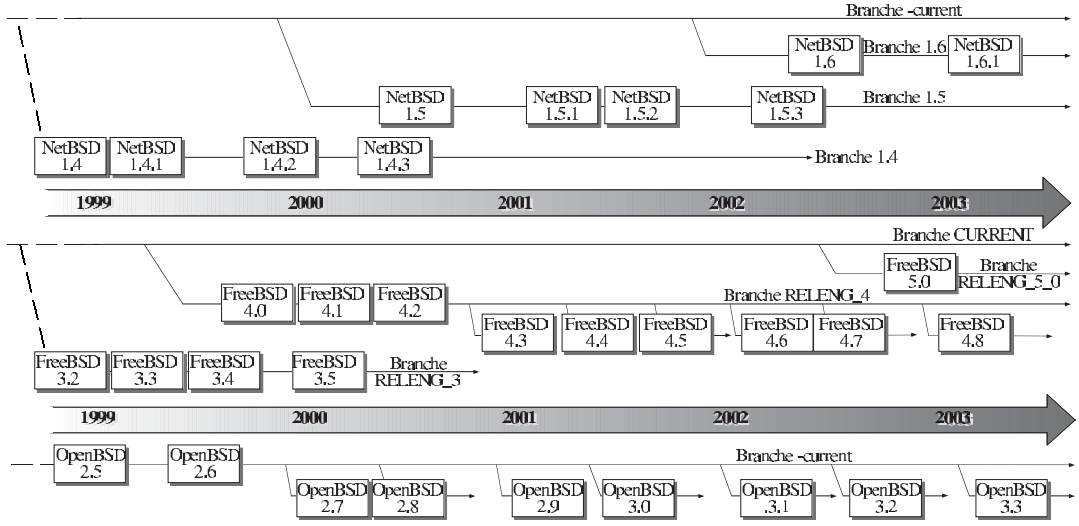


Figure 13–3 Branches de développement dans les projets BSD

Les méthodes de mise à niveau des sources dépendent du système : FreeBSD et OpenBSD fournissent des correctifs (*patches*), à appliquer avec la commande **patch** comme suit :

```
# cd /usr/src
# ftp ftp://ftp.FreeBSD.org/pub/FreeBSD/CERT/patches/SA-03:01/cvs.patch
# patch < cvs.patch
```

NetBSD recommande de mettre à jour l'arbre de sources. Exemple de mise à jour avec **cvs** :

```
# cd /usr/src/lib/libc
# cvs -d:pserver:anoncvs@anoncvs.fr.netbsd.org:/pub/NetBSD-CVS login
Logging in to :pserver:anoncvs@anoncvs.netbsd.org:2401/pub/NetBSD-CVS
CVS password: anoncvs
# cvs -d:pserver:anoncvs@anoncvs.fr.netbsd.org:/pub/NetBSD-CVS update -P -d
```

Signalons au passage la capacité de **cvs** à générer des correctifs au format de la commande **patch** :

```
# cvs -d:pserver:anoncvs@anoncvs.netbsd.org:/cvsroot diff -rnetbsd-1-6 -U4 > patchfile
# patch < patchfile
```

Une fois les sources à jour, il faut recompiler. On suivra les conseils de l'alerte de sécurité, car les différents éléments de l'arbre de sources se compilent différemment. Exemple extrait d'une alerte de sécurité de FreeBSD :

```
# cd /usr/src/gnu/usr.bin/cvs
# make obj && make depend && make && make install
```

ASTUCE **Simplifiez-vous la vie avec CVS**

La commande **cvs login** n'est nécessaire que la première fois ; le mot de passe reste ensuite stocké dans le fichier .cvspass du répertoire personnel.
On évitera la longue ligne de commande avec un alias dans le fichier .profile ou .bashrc. Exemple en **sh**, **ksh**, ou **bash** :

```
alias cvs='cvs -d:pserver:anoncvs@anoncvs.fr.netbsd.org:/pub/NetBSD-CVS'
```

Si on travaille avec plusieurs dépôts CVS, on mettra en place plusieurs alias. Par exemple **nbcvs** pour le dépôt du projet NetBSD et **fbcvs** pour celui de FreeBSD.

Configuration du noyau

Le cas du noyau est particulier, dans la mesure où on peut le configurer à la compilation. Cela tient au fait qu'au démarrage de la machine, il est intégralement chargé en mémoire, et y reste ensuite. Configurer le noyau permet d'éliminer les fonctions dont on n'a pas l'utilité pour améliorer les performances : tout code inutilisé mobilise en effet de la mémoire de manière permanente.

La désactivation de sous-systèmes inutiles évitera également certains problèmes de sécurité, rares mais pas inexistants dans le noyau.

Inversement, on souhaite parfois activer un sous-système ou un pilote jugé trop expérimental ou trop gourmand pour être activé par défaut. Les motivations pour configurer le noyau ne manquent donc pas. Voyons le cas des systèmes BSD.

B.A.-BA **Les patches**

Les correctifs (*patches*) sont de petits fichiers décrivant les modifications à apporter à un fichier texte, en fournissant une liste de lignes à supprimer ou ajouter. La commande **diff** les crée ; la commande **patch** les applique. Exemple de fragment de patch « contextuel » (rappelant quelques lignes de contexte) :

```
@@ -77,4 +78,6 @@
 void    usage(void);
+void    prthumanval(int64_t, char *);
+void    prthuman(struct statfs *, long);

-int    aflag, iflag, kflag, lflag;
+int    aflag, hflag, iflag, kflag, lflag;
 char   **typelist = NULL;
```

CULTURE **CVS, SUP, et CVSup**

CVS (*Concurrent Version System*) est un outil de contrôle de versions concurrentes, permettant à plusieurs programmeurs de travailler de concert sur le même dépôt de sources. Beaucoup d'équipes de développement y recourent, dont celles des trois BSD. La commande **cvs** permet de travailler avec un dépôt de sources CVS.
cvs permet de modifier les sources dans le dépôt, de consulter l'historique de leur modification, ou de les mettre à jour. Mais si l'on ne s'intéresse qu'à la mise à jour des sources, **cvs** est loin d'être l'outil le plus performant : il vaut mieux se tourner vers SUP et CVSup, deux outils conçus pour une distribution efficace des grandes arborescences de fichiers.
CVSup est plus efficace que SUP, mais il n'est pas disponible pour n'importe quel système Unix, car il est programmé en Modula-3. Le compilateur de Modula-3 utilisé pour compiler CVSup s'appelle **ezm3**, et, à l'heure où sont écrites ces lignes, il ne peut produire des binaires que pour i386, Alpha et Sparc.
Les trois projets BSD disposent de serveurs CVS, SUP et CVSup que l'on peut utiliser pour maintenir leurs sources à jour. NetBSD et OpenBSD mettent peu en avant CVSup, dans la mesure où ils supportent un large nombre de plateformes n'y ayant pas accès.

RAPPEL **Plateforme**

La plateforme est l'architecture matérielle de travail. Les trois BSD appellent « i386 » les compatibles PC. NetBSD et OpenBSD appellent « macppc » le PowerMacintosh. Selon leur génération, les stations Sun porteront le nom « sun2 », « sun3 », « sparc », ou « sparc64 ».

ASTUCE **/sys**

Les chemins sous /usr/src/sys sont un peu longs, mais /sys est un raccourci (lien symbolique) vers /usr/src/sys. On peut ainsi se contenter de /sys/arch/sparc64 pour signifier /usr/src/sys/arch/sparc64.

CULTURE **Noyau GENERIC**

Le noyau *GENERIC* est le noyau installé par défaut, incluant la prise en charge d'un maximum de matériels, systèmes de fichiers, et protocoles, mais excluant ce qui est encore trop expérimental. Il conviendra à tous les usages, mais ne sera probablement optimisé à aucune utilisation donnée.

ATTENTION **Pages man de pci**

Attention à ne pas confondre `pci(3)` et `pci(4)`.

SCÉNARIO CATASTROPHE **Échec de la compilation**

config peut être la cause d'erreurs de compilation d'un noyau plus récent que le système. Il faut alors le mettre à jour et recommencer, comme suit :

```
# cd /usr/src/usr/sbin/config
# make && make install
```

ASTUCE **Compiler sans être root**

Nul besoin d'être root pour compiler le noyau : il suffit de disposer des droits en écriture dans les répertoires conf et compile. Il faudra en revanche avoir les droits de l'administrateur pour installer le nouveau noyau.

Tout se déroule dans le répertoire concernant la plateforme de compilation. Par exemple, sous NetBSD ou OpenBSD pour une station Sun Ultra, il s'agira de /usr/src/sys/arch/sparc64.

Ce répertoire contient notamment un sous-répertoire conf, renfermant les fichiers de configuration du noyau, et un sous-répertoire compile, lieu de la compilation à proprement parler.

On pourra modifier les fichiers de configuration du noyau avec un éditeur de texte. En général, tout fichier personnalisé brode sur une copie de *GENERIC*. Exemple de fichier de configuration (fragments du *GENERIC* de NetBSD/macppc) :

```
# Standard system options
#options        UCONSOLE        # users can use TIOCCONS (for xconsole)
#options        INSECURE        # disable kernel security levels

options         RTC_OFFSET=0    # hardware clock is this many mins. west of GMT
options         NTP             # NTP phase/frequency locked loop
options         KTRACE          # system call tracing via ktrace(1)
options         SYSTRACE        # system call vetting via systrace(1)
(...)
cpu*    at mainbus?
bandit* at mainbus?
grackle* at mainbus?
uninorth* at mainbus?

pci*    at bandit? bus ?
pci*    at grackle? bus ?
pci*    at uninorth? bus ?
pci*    at ppb? bus ?
(...)
pseudo-device   md              1       # memory disk device
pseudo-device   loop                    # network loopback
pseudo-device   bpfilter        8       # packet filter
pseudo-device   ipfilter                # IP filter (firewall) and NAT
```

On y trouve trois parties principales : les déclarations d'options, la liste des pilotes et leurs points d'attache, et la liste des pseudo-périphériques (*pseudo-devices*). Chaque nom de pilote ou de pseudo-périphérique dispose en principe d'une page **man** en section 4 détaillant son rôle. Exemple : `pci(4)`.

Les options spécifiées au début du fichier sont documentées sous NetBSD et OpenBSD dans la page **man** `options(4)`. Pour FreeBSD 5.0 et versions suivantes, reportez-vous aux documents /sys/conf/NOTES et /sys/i386/conf/NOTES. Pour les version antérieures à FreeBSD 5.0, le détail des options se trouve dans /sys/i386/conf/LINT.

La configuration prête, on crée l'environnement de compilation adapté puis on compile. La démarche diffère légèrement selon que l'on utilise FreeBSD ou ses deux cousins NetBSD et OpenBSD.

Sur NetBSD et OpenBSD, c'est la commande **config** qui crée l'environnement de compilation. Si le fichier de configuration s'appelle KERN, on procédera comme suit :

```
$ pwd
/sys/arch/sparc64/conf
$ config KERN
Don't forget to run "make depend"
$ cd ../compile/KERN
$ make depend && make
```

Le répertoire KERN, créé par la commande **config**, contient tous les Makefile et fichiers .h nécessaires à la compilation. On compile alors en tapant **make depend && make**

Sous FreeBSD, la méthode recommandée diffère légèrement (dans notre exemple le fichier de configuration s'appelle toujours KERN) :

```
$ pwd
/usr/src/sys/arch/i386/conf
$ cd /usr/src
$ make buildkernel KERNCONF=KERN
```

Installation du nouveau noyau

La compilation produit un lot de fichiers objet .o et un fichier exécutable : le noyau. Nommé kernel sous FreeBSD, NetBSD le baptise netbsd, et OpenBSD, bsd. Sous NetBSD et OpenBSD, on installera un nouveau noyau en le copiant à la racine du système de fichiers et en redémarrant la machine.

Le noyau de FreeBSD utilise couramment de nombreux modules, ce qui rend son installation légèrement différente : la méthode recommandée est de taper **make installkernel** dans /usr/src. Dans les versions antérieures à la 5.0, cette commande copie le noyau à la racine et les modules dans le répertoire /modules. L'ancien noyau est appelé /kernel.old pour y revenir facilement si le nouveau noyau ne fonctionne pas correctement.

À partir de FreeBSD 5.0, le noyau et les modules qu'on lui a compilés sont placés dans un sous-répertoire de /boot. L'objectif est de permettre la cohabitation de versions différentes du noyau et des modules correspondants. Par défaut, **make installkernel** place noyau et modules dans /boot/kernel, mais on pourra indiquer un autre emplacement comme suit : **make install KODIR=/boot/newkernel**

```
$ ls /boot/kernel
3dfx.ko                 if_sis.ko               nsp.ko
aac.ko                  if_sk.ko                ntfs.ko
aac_linux.ko            if_sl.ko                nullfs.ko
accf_data.ko            if_sn.ko                nwfs.ko
(...)
```

B.A.-BA **Noyau modulaire**

Très populaires dans le monde GNU/Linux et sous FreeBSD, les modules noyau permettent de charger et décharger à chaud certaines fonctionnalités : couches de compatibilité binaire, systèmes de fichiers, pilotes de matériel… Ils permettent au noyau d'évoluer sans devoir redémarrer la machine. Sous FreeBSD, certains modules sont chargés explicitement avec la commande **kldload**, d'autres sont appelés automatiquement en cas de besoin.
NetBSD et OpenBSD disposent aussi de la capacité de charger des modules dans le noyau, mais utilisent très peu cette fonctionnalité.

CAS PARTICULIERS **Si on fait du netboot**

Le noyau n'est pas toujours à la racine du système de fichiers : en *netboot*, il se trouve sur un serveur de fichiers ; en cas de démarrage depuis un autre système d'exploitation (cas de MacOS sur les mac68k ou de Windows CE sur les assistants personnels), il est sur une autre partition.
Dans ces situations, il faut évidemment remplacer l'ancien noyau par le nouveau.

PIÈGE À C… **Ne jetez pas trop vite l'ancien noyau**

Lors de l'installation d'un nouveau noyau, il est prudent de conserver l'ancien quelques temps, sous un autre nom. Si le nouveau noyau n'est pas capable de démarrer la machine (comme suite à une suppression distraite de la prise en charge du contrôleur de disque), l'ancien noyau permettra de remettre rapidement la machine en service (pour savoir comment choisir le noyau invoqué au démarrage de la machine, reportez-vous au chapitre 5).

ATTENTION **Noyau et modules**

Les modules et le noyau doivent toujours être compilés avec les mêmes sources et les mêmes options de compilation, sous peine d'obtenir un système instable, les modules n'étant pas adaptés au noyau. En utilisant **make installkernel**, de tels problèmes ne devraient pas apparaître, mais soyez vigilant si vous copiez le noyau ou les modules vous-même.

Recompilation du système entier

On souhaite parfois recompiler tout le système, par exemple pour suivre la branche *-current*. Dans ce cas, on se placera au sommet de l'arbre de sources, dans /usr/src. On tapera alors **`make buildworld`** sous FreeBSD, **`make build`** sous OpenBSD, et **`./build.sh build`** sur NetBSD.

La compilation de l'ensemble du système est une opération complexe. Il existe parfois des dépendances cycliques sur les outils de compilation eux-mêmes, provoquant des erreurs s'ils sont trop anciens. OpenBSD propose des quantités considérables de documentation sur le Web pour parer au pire : http://www.openbsd.org/faq/upgrade-minifaq.html. Une section du *Handbook* de FreeBSD traite du sujet : http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/makeworld.html. FreeBSD propose encore un fichier UPDATING à la racine de l'arbre de sources, où sont consignés les problèmes potentiels et leurs solutions.

Compilation croisée

NetBSD dispose également d'un fichier UPDATING, mais sa compilation s'est considérablement simplifiée depuis NetBSD 1.6, avec l'introduction d'une infrastructure de compilation où les outils de compilation sont séparés du reste. Le système de compilation fait une première passe en compilant les outils de compilation, puis les utilise pour compiler le système lui-même. Tout se fait en invoquant le script shell **`build.sh`** situé à la racine de l'arbre de sources.

Cette infrastructure fait plus encore : elle permet notamment la compilation croisée de n'importe quel port de NetBSD depuis n'importe quel autre port, et même, en théorie, depuis n'importe quel système POSIX disposant d'un environnement de compilation **`gcc`**.

Pour compiler une distribution de NetBSD/mac68k depuis n'importe quel NetBSD, on peut ainsi taper :

```
$ cd /usr/src
$ ./build.sh -m mac68k
```

Ceci fonctionnera aussi depuis FreeBSD, GNU/Linux, Solaris... certains ont même poussé le vice jusqu'à compiler NetBSD depuis Windows, avec l'environnement Cygwin.

La compilation croisée est très intéressante : un mac68k met plusieurs jours à compiler tout le système, alors que cette opération prend moins d'une heure à une machine récente.

L'option **`-t`** de **`build.sh`** ne lui fera compiler que les outils de compilation, ce qui peut servir à ne compiler qu'un morceau de l'arbre. Les outils de compilation sont compilés sous /usr/src/tools/obj. Il suffira ensuite d'utiliser **`nbmake-mac68k`** pour compiler des binaires mac68k.

CULTURE **Cygwin**

L'environnement Cygwin permet de tirer parti des capacités POSIX de Windows qui furent introduites à partir de NT. On peut ainsi installer et utiliser **`gcc`** ainsi que beaucoup d'autres logiciels libres.

Compilation automatique

Le système de compilation de NetBSD est utilisé pour compiler en boucle tous les ports de NetBSD sur une machine actuellement nommée tgm (*The Great Machine*, ou la formidable machine). Elle fournit chaque jour une distribution binaire pour chaque plateforme supportée de NetBSD-*current* et de la branche stable. Le résultat est accessible à l'adresse suivante : http://releng.netbsd.org. On peut donc, si on le souhaite, s'approvisionner sur cette machine pour faire ses mises à jour binaires.

FreeBSD fournit un service similaire sur http://current.freebsd.org. Les architectures i386 et alpha sont compilées en boucle pour les différentes branches du projet.

Tout ceci nous mène donc aux problèmes de mise à jour des binaires.

Mise à jour des binaires

On dispose parfois de mises à jour binaires, issues d'un serveur spécialisé ou compilées par vos soins avec l'intention de les installer sur un parc de machines identiques. On évite l'étape de compilation : il suffit de remplacer les binaires avec ces versions plus récentes, mais il y a des écueils à éviter.

Ce qui ne change pas : il faudra penser à redémarrer tout *daemon* mis à jour et remplacer tous binaire statique lié à toute bibliothèque remplacée.

Nouvelle difficulté ; s'assurer d'installer des binaires intégrant le correctif. On peut facilement vérifier cela dans des sources, mais rien n'est plus tentant que d'accorder une confiance aveugle à un binaire en rien corrigé : c'est bien plus difficile à distinguer.

Une solution consiste à contrôler les balises RCS dans les fichiers binaires. Chaque fichier dans l'arbre de sources en contient, régulièrement mises à jour par CVS, pour indiquer notamment le numéro de version et la date de modification. Elles sont embarquées dans le binaire avec une macro **__RCSID** telle que :

```
__RCSID("$NetBSD: ls.c,v 1.45 2002/09/27 12:01:51 simonb Exp $");
```

La commande **ident** permet d'extraire les balises et de savoir ce qui a été compilé dans le binaire. Exemple :

```
$ ident /bin/ls
/bin/ls:
     $NetBSD: crt0.c,v 1.23 2002/07/29 21:54:35 matt Exp $
     $NetBSD: cmp.c,v 1.16 2000/07/29 03:46:14 lukem Exp $
     $NetBSD: ls.c,v 1.45 2002/09/27 12:01:51 simonb Exp $
     $NetBSD: main.c,v 1.2 2000/07/29 03:46:15 lukem Exp $
     $NetBSD: print.c,v 1.33 2002/11/09 12:27:08 enami Exp $
     $NetBSD: stat_flags.c,v 1.15 2002/11/16 13:42:36 itojun Exp $
     $NetBSD: util.c,v 1.23 2002/11/09 12:27:08 enami Exp $
```

On s'assurera de bien corriger un problème de sécurité en connaissant les fichiers concernés et les numéros de version correspondant à l'application du *patch* de sécurité. **ident** permet de contrôler cela facilement.

PLUS LOIN **Mise à jour automatisée des binaires**

Le projet FreeBSD met au point un système de mise à jour des binaires appelé **binup**, encore en développement à l'heure où sont écrites ces lignes. On trouvera des informations à ce sujet à l'adresse http://www.freebsd.org/projects/updater.html.

CULTURE **RCS**

RCS est un ancêtre de CVS. Il est disponible dans le système de base à travers les commandes **ci** et **co**. RCS utilise des balises pour indiquer les numéros de version (*revision*) des fichiers, reprises à l'identique par CVS.

Il est facile sous FreeBSD de connaître le numéro de version de chaque fichier incorporant les corrections de sécurité car les alertes de sécurité contiennent ces informations :

```
Branch                                                          Revision
  Patch
- ----------------------------------------------------------------------
RELENG_4
  src/crypto/openssl/crypto/rsa/rsa_eay.c                        1.2.4.6
  src/crypto/openssl/crypto/rsa/rsa_lib.c                        1.2.2.7
  src/crypto/openssl/ssl/s3_srvr.c                             1.1.1.1.2.7
```

Pour NetBSD et OpenBSD, on trouvera cette information dans le CVSWeb, respectivement aux adresses http://cvsweb.netbsd.org, et http://www.openbsd.org/cgi-bin/cvsweb/. Il faut connaître les fichiers concernés, et contrôler leurs historiques.

Cette méthode est inapplicable aux programmes compilés sans embarquer les numéros de version des fichiers : on se reposera sur leur date de création pour tenter de deviner les mises à jour qu'ils comportent.

PLUS LOIN **CVS et CVSWeb**

CVS est un outil de contrôle de version des sources permettant à de nombreux développeurs de travailler sur le même projet. Il utilise un dépôt de sources, où chaque fichier reçoit un numéro de version (*revision*), et où chaque modification de fichier est motivée dans un message.

CVSWeb est un frontal Web pour explorer le dépôt de sources. On peut l'utiliser pour retrouver dans les journaux de modification d'un fichier le numéro de version intégrant une correction de problème.

SUR LES AUTRES UNIX **Balises RCS**

Ces balises (*tags*) RCS peuvent servir à bien d'autres usages, comme découvrir l'origine des programmes utilisés dans MacOS X (ici MacOS X.2).

```
$ ident /usr/bin/* 2> /dev/null | grep FreeBSD | wc -l
      99
$ ident /usr/bin/* 2> /dev/null | grep NetBSD | wc -l
     242
$ ident /usr/bin/* 2> /dev/null | grep OpenBSD | wc -l
     218
$ uname -a
Darwin sonja.local. 6.5 Darwin Kernel Version 6.5: Mon Apr  7 17:05:38 PDT 2003;
 root:xnu/xnu-344.32.obj~1/RELEASE_PPC  Power Macintosh powerpc
```

Où l'on découvre que MacOS X.2 a beaucoup puisé dans les trois BSD, et plutôt dans NetBSD et OpenBSD, contrairement au discours officiel d'Apple qui ne mentionne que l'héritage de FreeBSD. Cela peut irriter des partisans des autres systèmes, dont l'auteur.

Index