1 Le concept de processus

Un processus est un programme en train de tourner sur l'ordinateur considéré. Il est identifié par un numéro, son PID (Process IDentification), et les numéros partent de 1 au moment où la machine démarre. Le processus `init` est donc identifié par le numéro 1.
Un processus peut lancer d'autres processus, appelés processus *fils*. En fait, tous les processus autres que `init` sont lancés par un processus « père » et cela donne donc lieu à une arborescence de processus, dont `init` est la racine. `init` est quant-à lui lancé au boot de la machine. Les shells sont en général les nœuds ayant le plus de fils différents au cours du temps.

Un processus agit toujours au nom d'un unique utilisateur du système *et il en a tous les droits*. Cela a pour conséquence immédiate qu'il ne faut lancer un processus que si l'on a toute confiance en ce qu'il fait ! surtout si l'on est `root`...

> La possibilité de savoir en détail ce que fait chaque programme qu'on utilise devrait être *un droit inaliénable* pour chaque citoyen car les actions des processus qui travaillent en son nom *engagent sa responsabilité*. Cela implique de pouvoir lire les sources et les spécifications de tout logiciel. Les logiciels qui respectent ce droit son dit « open source » ; cela ne signifie pas pour autant que l'usage du logiciel soit gratuit.
>
> Naturellement peu de gens ont le loisir de lire et comprendre toutes les sources de tous les logiciels qu'ils utilisent, cependant les sources d'un logiciel open source restent, au bilan, lues par de nombreuses personnes dans le monde et c'est ce « contrôle collectif » qui garantit les fonctionnalités exactes du logiciel à tous les utilisateurs.
>
> De manière surprenante de nombreux logiciels, non seulement n'offrent pas ces garanties, mais contiennent et exécutent de plus des fonctions non documentées. Ne soyez pas naïfs ni angéliques : n'utilisez ces logiciels qu'avec la plus grande méfiance et utilisez toujours des versions open source lorsqu'elles existent.
> L'exécution de fonctions non documentées pour « pister » toutes vos actions est particulièrement fréquente sur les smartphones par exemple, ou encore sur la plupart des pages web que l'on visite, dont les moteurs de recherche les plus connus. Surveillez sévèrement les scripts exécutés lorsque vous surfez sur internet, par exemple en installant `noscript` sur votre navigateur.

Un processus possède toujours au moins les 3 canaux standard suivants :
— une entrée standard
— une sortie standard
— une sortie d'erreurs
Les shells savent manipuler ces trois canaux pour tous les processus qu'ils lancent. Par défaut, l'entrée standard est le clavier, la sortie standard est la fenêtre du terminal et la sortie d'erreur est également dirigée vers le terminal. Il est possible de les *rediriger* :
— On peut par exemple utiliser le contenu d'un fichier comme entrée standard d'un processus
`$ commande < fichierEnEntree`
— On peut créer un fichier contenant la sortie standard d'un processus (écrase le fichier s'il existait déjà)
`$ commande > fichierResultat`
On peut aussi ajouter à la fin d'un fichier existant
`$ commande >> fichierResultat`
— Enfin on peut enchaîner les commandes en fournissant la sortie d'une commande en entrée d'une autre. On dit alors qu'on « pipe » les commandes
`$ commande1 | commande2 | commande3`

La commande `ps` permet de lister les processus qu'on a lancés depuis le shell et qui tournent au moment du lancement de la commande. Cette commande admet plusieurs options utiles, dont « `-x` » qui permet de voir *tous* les processus qui tournent en votre nom (pas seulement les descendants du shell), et « `-aux` » qui liste tous les processus qui tournent sur la machine, quels qu'en soient les propriétaires (il y en a beaucoup, donc c'est une bonne idée de « piper » la sortie standard de `ps` dans l'entrée standard de `less` afin de paginer le résultat).

La commande `top` est similaire à « `ps -aux` » sauf qu'elle ne montre sur une page que les processus les plus gourmands en puissance de calcul. La page est mise à jour en temps réel et on sort de la commande en tapant `q` dans son entrée standard. Très utile lorsqu'on se demande pourquoi la machine a un trou de performance...

Enfin la commande `kill` tente de tuer les processus dont on lui donne les numéros en arguments. Elle « tente » en ce sens qu'elle fait passer un message à ces processus qui leur demande de terminer. Un processus parti dans un bug quelconque peut très bien ignorer cette demande, de même qu'un processus qui s'estime dans une « section critique »

et refuse de sortir afin de préserver par exemple la cohérence de ses données. Lorsqu'un processus termine (en fin de programme ou par un `kill`), il doit rattacher ses processus fils à son propre père pour ne pas casser l'arbre des processus. Si l'un de ses fils ne répond pas, il doit donc l'attendre avant terminer. L'option `-9` est plus violente : `kill -9` envoie un signal (très) impératif au processus et l'arrête net. Dans ce cas, tous les processus fils qui ne répondent pas asez vite ne seront pas rattachés à l'arborescence des processus et on les nomme des *processus zombies*. Ces zombies tournent souvent « dans le vide » et peuvent dégrader les performances d'une machine, mieux vaut les tuer aussi (repérés par un Z avec la commande `ps`).

2 Les processus « shell »

Un shell est un processus dont le rôle est par défaut d'attendre que l'utilisateur tape une commande au clavier, d'interpréter la ligne qu'il a tapée et de lancer le ou les processus qu'implique cette commande. Pour le shell comme pour tout processus, lorsqu'il lance d'autres processus, ce sont ses processus *fils*.

> Dans la première phrase du paragraphe précédent, au vu de ce qu'on a expliqué sur les processus, vous aurez compris qu'il faut lire « qu'une ligne apparaisse sur son entrée standard » à la place de « que l'utilisateur tape une commande au clavier », car le shell est un processus comme un autre. Cela permet donc de donner au shell un fichier préparé à l'avance en entrée standard, réalisant une combinaison de commandes d'une complexité aussi grande que l'on souhaite.

Comme on l'a vu, lorsqu'un utilisateur se logue avec succès, c'est généralement un shell qui est lancé pour l'accueillir sur la machine (selon ce qui est défini dans `/etc/passwd`). Ce n'est pas le seul usage d'un shell, par exemple lorsque vous ouvrez une fenêtre « de terminal » (qui simule les vieux terminaux informatiques de l'époque où les interfaces graphiques n'existaient pas), vous obtenez une fenêtre dans laquelle tourne un shell, qui vous permet donc de lancer de nombreuses commandes, avec des finesses de choix des options qui sont totalement inaccessibles *via* les menus d'une interface graphique. Pour gérer un système informatique, on utilise donc intensivement les fenêtres de terminal. Lorsqu'on se logue avec une interface graphique, on doit considérer que c'est le shell « de login », dont l'entrée standard n'est plus votre clavier mais un fichier prédéfini, qui lance tous les processus qui vont gérer l'interface graphique pour l'utilisateur. Il existe plusieurs programmes qui sont des shells et nous n'utiliserons ici que le plus courant qui est `bash`[1].

Lorsqu'un processus shell est lancé, il dispose de plusieurs variables dont les valeurs évitent d'aller chercher dans les fichiers de configuration les informations dont on se sert souvent. Par exemple :
- `USER` contient le nom de login du propriétaire du processus shell. Cela évite de parcourir `/etc/passwd` en cherchant la ligne qui correspond à l'identifiant (l'UID) du propriétaire du processus. Dans un shell, on obtient la valeur d'une variable en mettant un « $ » devant, ainsi la commande « `echo $USER` » depuis votre shell fournit votre nom de login.
- `HOME` contient l'adresse absolue de votre répertoire personnel.
- `LANG` contient votre langue préférée sous une forme codifiée compréhensible par le système (souvent `FR` pour le français) et cette variable est exploitée pour formater la sortie de certaines commandes (par exemple `ls` suit ainsi l'ordre alphabétique du langage de l'utilisateur).
- `HOSTNAME` contient le nom de la machine sur laquelle tourne le processus.
- `DISPLAY` contient un identifiant de l'écran graphique qu'il faut utiliser pour afficher les résultats d'une commande *via* l'interface graphique (note : ceci est indépendant de la sortie standard du processus).
- `PRINTER` contient le nom de votre imprimante préférée.
- *etc.*

La variable `PATH` mérite à elle seule un paragraphe d'explications :

Lorsqu'un shell doit lancer un processus à la suite d'une commande tapée par l'utilisateur, par exemple « `man ls` » :
- le shell doit d'abord trouver le fichier exécutable qui contient le code du processus à lancer, pour notre exemple le fichier `/usr/bin/man`,
- ensuite il le charge en mémoire et indique au système qu'il faut l'exécuter avec les bons arguments, pour notre exemple l'argument « `ls` ».
- Par défaut le shell « attend » que le processus soit terminé avant de réafficher le prompt en attente d'une nouvelle commande, toutefois il est possible de lui dire de ne pas attendre la fin de la commande en ajoutant un « `&` » à la fin de la ligne de commande.

Revenons à la recherche du fichier exécutable d'une commande. Si la commande avait été « `su` », le shell aurait dû trouver le fichier à l'adresse `/bin/su`, donc dans un répertoire différent. Pour la commande « `openarena` » le shell aurait lancé `/usr/games/openarena`, donc un troisième répertoire... Compte tenu du nombre de fichiers présents dans

1. pour *Bourne-again shell*, jeu de mot avec *born again* parce que `bash` est une extension du shell « historique » de linux (`sh`) qui avait été programmé par Stephen Bourne

l'arborescence du système, il serait impensable de parcourir tout l'arbre des fichiers pour trouver chaque commande et c'est en fait la variable `PATH` qui contient la liste des répertoires que le shell va explorer pour trouver les commandes de l'utilisateur. Dans cette variable, les répertoires sont séparés par des « : ». Par exemple :

```
$ echo $PATH
/usr/bin:/bin:/usr/local/bin:/usr/games:/usr/lib/qt4/bin:/usr/lib/qt5/bin:/home/bioinfo/bernot/bin
```

La commande `which` permet de savoir où le shell trouve une commande donnée ; par exemple « `which man` » retourne `/usr/bin/man`.

> Au passage, si vous administrez un système et qu'un utilisateur vous demande une petite réparation qui nécessite de passer `root`, tapez « `/bin/su` » car si vous tapez seulement « `su` » l'utilisateur peut très bien avoir un `$PATH` qui vous fait exécuter une commande `su` qui n'est pas celle que vous croyez, et il peut ainsi récupérer le mot de passe de `root`... En fait, par principe, ne passez `root` qu'à partir de votre propre compte utilisateur.

Les variables mentionnées jusqu'ici sont dites « d'environnement » c'est-à-dire que lorsque le shell lance une commande, il transmet la valeur de ces variables à ses processus fils. Si vous définissez vous même une variable (par exemple « `LIEU=Sophi@Tech` »), elle est par défaut « locale ». Pour la transmettre aux processus fils, il faut la rendre globale, c'est-à-dire d'environnement, avec la commande `export` (par exemple « `export LIEU` »). Par convention, les variables globales portent généralement des noms ne contenant que des majuscules.

Un shell ne se contente pas seulement de lancer des processus tapés par l'utilisateur sur une seule ligne, c'est aussi un langage de programmation impératif qui possède donc à ce titre des primitives telles que `if`, `while`, `for`, `case`, la gestion des variables, les redirections des entrées et sorties des processus, *etc.* En revanche, les shells sont indigents en termes de structures de données. Tout est chaîne de caractère !

On peut donc non seulement taper des commandes assez sophistiquées, mais aussi écrire des programmes dans des fichiers et les faire interpréter par le shell comme s'il s'agissait de son entrée standard.

Un cas particulier important de tels fichiers sont les fichiers d'initialisation :
- Lorsqu'on lance un `bash`, avant de donner la main à l'utilisateur il exécute le fichier `$HOME/.bashrc` s'il existe. Ceci permet de personnaliser le shell. Par exemple on peut y placer les « *alias* » : lorsqu'on exécute la ligne « `alias ll='ls -l'` » alors il suffira ensuite de taper `ll` au lieu de `ls -l` pour obtenir les informations « longues » sur les fichiers.
- Lorsque le shell est lancé directement par un login de l'utilisateur, il exécute d'abord (donc avant le `.bashrc`) le fichier `$HOME/.profile` s'il existe [2]. Ceci permet typiquement d'initialiser de manière personnelles les variables globales. Par exemple étendre `$PATH` avec un répertoire personnel avec la ligne « `PATH=$HOME/bin:$PATH` » suivie de « `export PATH` ».

Pour terminer cette section, mentionnons que par convention le caractère Ctrl D en début de ligne indique la fin de l'entrée standard d'un processus. Ainsi pour sortir d'un shell, il suffit de taper ce caractère au lieu d'une nouvelle commande ; le shell comprend qu'il ne recevra pas d'autres commandes et il s'arrête donc. On obtient le même résultat avec la commande `exit`. Les shells « de login » (ceux lancés par un login de l'utilisateur) font exception pour éviter de se déloguer intempestivement sur une simple faute de frappe. Il faut utiliser la commande `logout` pour sortir de ces shells là, et donc se déloguer.

3 Les environnements graphiques

Lorsqu'on se connecte en étant physiquement présent au clavier d'un ordinateur, on bénéficie généralement d'une *interface graphique* à l'écran de cet ordinateur. La plupart du temps, l'interface par défaut pour le login des utilisateurs est alors elle-même graphique et, après un login en succès, l'utilisateur n'est pas face à un shell mais face à un *environnement graphique*. Un tel environnement est fondé sur la notion de « fenêtre » et la seule fenêtre qui soit obligatoirement présente est le fond d'écran, appelée *root window*. Cette fenêtre de fond d'écran est gérée par un processus qui surveille les actions de la souris (ou du clavier) dans ce fond d'écran, redessine le fond d'écran lorsqu'une autre fenêtre est déplacée ou supprimée, *etc.* C'est en quelque sorte le processus « père » de l'environnement graphique. Il est également conçu pour lancer des processus fils qui gèrent le cas échéant :
- des fenêtres « standard » ; à chaque fenêtre est attaché un processus qui en gère le contenu ; par exemple `firefox` lorsque vous lancez un navigateur web, ou `xterm` ou ses variantes lorsque vous lancez une fenêtre de terminal (ce dernier processus, outre la gestion graphique de la fenêtre, ayant pour fonction principale de lancer un shell) ...

2. `bash` teste aussi si `$HOME/.bashprofile` existe.

— des « icônes » ; il s'agit simplement d'une fenêtre de petites taille à laquelle est associé un lien symbolique. Un sous-répertoire de votre homedir est le plus souvent dédié à ces liens symboliques et plus généralement les fils de ce répertoire sont ceux qui donnent lieu à des icônes visibles à l'écran (répertoire souvent appelé « le bureau », « desktop »). Si le lien pointe sur un exécutable, il est lancé sur demande à la souris, et s'il pointe sur un autre type de fichier alors une liste intermédiaire est consultée qui indique pour chaque type de fichier quel exécutable doit être lancé pour le gérer (liste des « mime types »).

    Chaque installation de « package » qui introduit un nouveau type de fichier de données ou un nouveau logiciel comprends une modification de la liste des mime types. `root` peut modifier les priorités par défaut de cette liste (par exemple il y a de nombreux lecteurs de vidéos, le dernier installé se place généralement en premier sur la liste des mime types et cet ordre d'installation n'est pas forcément judicieux) et chaque utilisateur peut faire de même dans les paramètres de son environnement graphique.

— des « tableaux de bord » ou « barres d'outils » ou « panels » qui sont des fenêtres de grande longueur (souvent tout l'écran) et de faible largeur (généralement celle des icônes) qui contiennent elles-mêmes des icônes permettant de lancer d'autres fenêtres ou processus utiles ;
— des menus, qui peuvent être lancés depuis un tableau de bord ou par un clic de souris dans la root window, ou encore par un clic dans certaines parties des autres fenêtres : ils servent eux-mêmes à lancer de nouveaux processus et/ou nouvelles fenêtres.

De plus, le shell de login lance un processus « gestionnaire de fenêtres » (window manager). Le rôle de ce processus est, comme son nom l'indique, de placer, déplacer, iconifier ou changer de taille les fenêtres contenues dans la root window. La plupart du temps le gestionnaire de fenêtres permet d'effectuer ces actions à la souris en dessinant un *cadre* autour de chaque fenêtre (qu'on peut « attraper » avec la souris) et peut ajouter un bandeau au-dessus de la fenêtre pour inscrire le nom de la fenêtre et faciliter ces diverses opérations.

Tous ces éléments peuvent être paramétrés à la guise de l'utilisateur : couleurs, effets des boutons ou des touches, apparence des fenêtres et des icônes, processus préférés, nombre de tableaux de bord, menus et services offerts par les tableaux de bord, contenu des menus, ordre personnalisé de la liste des mime types, *etc.*

4 Services : les processus « démons »

Le gestionnaire de fenêtres, l'affichage de la root window, la mise à disposition des panels et certaines autres fenêtres ont ceci de particulier que ce sont des processus toujours présent qui attendent qu'on les « réveille » et les utilise (par exemple en passant ou cliquant dessus avec la souris). Il s'agit donc de processus qui ont été lancés par le shell de login et qui surveillent certains paramètres (typiquement la position de la souris, l'heure qu'il est et la mise à jour de l'horloge du tableau de bord, ou autre) pour déclencher diverses actions ou autres processus. De tels processus qui passent leur temps à attendre en surveillant quelques paramètres sont appelés des « processus démons » ou « services ».

De très nombreux démons tournent sur une machine. Parmi les plus courants, on peut citer ceux qui surveillent l'insertion d'un DVD, d'une clef USB, l'état du réseau, l'arrivée de mails, l'horloge du tableau de bord, *etc.* Certains sont lancés en votre nom (comme ceux qu'on a vus pour l'interface graphique), d'autres par `root` pour le système (limite de remplissage des disques, gestion des imprimantes, du son, recherche de mises à jour, défenses anti-piratage « murs de feu », *etc*). Les processus démons lancés par `root` (ou par des utilisateurs virtuels annexes) et dont le rôle est de faciliter la bonne marche du système complet sont aussi appelés des « services ».

L'un des rôles du super-user `root` est de choisir un bon compromis entre, d'une part, les facilités d'utilisation apportées par ces services, et d'autre part, les performances du système (car une multitude de démons finit par occuper le processeur de façon sensible) et la sécurité du système : certains services « ouvrent » des accès au monde extérieur, comme ssh, Apache ou Skype, qui sont autant de voies d'attaques pour les pirates, d'autres services renforcent la sécurité en surveillant ces mêmes voies. Il faut toujours éviter d'ouvrir des services qui ne sont pas indispensables aux utilisateurs, et si un service ouvre une voie extérieure, il faut (1) que le firewall (« murs de feu ») la surveille et (2) jeter régulièrement un oeil aux journaux systèmes générés par le firewall pour repérer les tentatives de piratages trop fréquentes : un service réseau ouvert est généralement l'objet de plusieurs dizaines d'attaques par jour, par des robots qui parcourent automatiquement les adresses IP.

5 Administration du réseau et sécurité du système

5.1 Principes du réseau internet

Lorsqu'un ordinateur doit envoyer des informations à un autre ordinateur *via* le réseau, il transmet les données par sa carte réseau et elles sont découpées en « paquets » de petite taille.

— Un paquet commence par une « en-tête » qui contient entre autres un numéro qui identifie la machine qui expédie le paquet, un numéro qui identifie la machine destinataire du paquet [3], le type des données transmises, *etc.*
Ensuite le « corps » du paquet contient des données et il faut généralement un bon nombre de paquets pour transférer toutes les informations lors d'une communication entre ordinateurs.

— La carte réseau (souvent intégrée à la « carte mère » de l'ordinateur) se charge de transformer chaque paquet en modulations d'intensité sur le cable réseau ou sur la wifi. Par défaut, *le paquet est transmis sans distinction à toutes les machines du réseau.* Toutes les machines lisent l'en-tête du paquet et si elles se reconnaissent en le numéro du destinataire (ou s'il s'agit d'un broadcast) alors le *processus démon* qui surveille la carte réseau va prendre le paquet en considération.
N'oubliez jamais par conséquent qu'une machine gérée de manière malveillante a toujours la possibilité de lire tous les paquets du (sous-)réseau auquel elle est connectée afin de récupérer les données qui transitent, informations confidentielles, mots de passe, *etc.*

Du fait que chaque paquet est transmis par une carte réseau à la totalité des machines connectées au réseau, il y a souvent des *collisions* : plusieurs machines qui émettent en même temps produisent sur le réseau un signal illisible. En cas de collision, les cartes réseau concernées ré-émettent leur paquet après un délai aléatoire. Il est donc matériellement impossible d'avoir un grand nombre de machines connectées à un même réseau car il y aurait trop de collisions. Pour cette raison, le réseau internet mondial est découpé en sous-réseaux (e.g. « `.fr` », « `.com` », « `.net` », « `.uk` », ...), ces sous-réseaux sont eux-mêmes découpés en sous-réseaux (e.g. « `univ-cotedazur.fr` », « `free.fr` », « `cnrs.fr` », « `genopole.fr` », ...), qui peuvent à leur tour être découpés (« `www.univ-cotedazur.fr` », « `bibliotheque.univ-cotedazur.fr` », « `ipmc.unice.fr` », « `i3s.unice.fr` », ...), et l'on peut continuer à découper autant que l'on souhaite. On obtient ainsi une arborescence de réseaux.

Chaque sous-réseau possède une *passerelle* : c'est un ordinateur qui porte 2 cartes réseau, l'une connectée au sous-réseau (par exemple `sophia.bibliotheque.univ-cotedazur.fr`) et l'autre connectée au réseau père (dans notre exemple, `bibliotheque.univ-cotedazur.fr`). La passerelle surveille tous les paquets qui passent dans son sous-réseau et lorsque l'adresse de la machine destinataire n'appartient pas au sous-réseau, le paquet est recopié sur la carte du réseau père. Inversement, la passerelle surveille tous les paquets du réseau père et si la machine destinaraire appartient à son sous-réseau local alors elle le recopie sur la carte du réseau local.

Il y a également des annuaires sur chaque sous-réseau (appelés DNS pour Domain Name Server) qui assurent la traduction entre le nom « humain » d'un sous-réseau ou d'une machine (comme `bibliotheque.univ-cotedazur.fr` par exemple) et son numéro « IP » utilisé dans les en-têtes de paquets (par exemple `85.118.46.110`). Avant d'envoyer ses paquets, un ordinateur demande au DNS le numéro IP qu'il mettra en en-tête du paquet [4].

Lorsqu'une machine démarre (« boote »), elle commence par envoyer des broadcasts pour connaître le numéro de la passerelle, du DNS, et de divers autres serveurs utiles. Ces serveurs peuvent répondre à ces demandes en fournissant chacun leurs caractéristiques mais en général un sous-réseau possède un serveur DHCP (Dynamic Host Configuration Protocol) dont le rôle est de centraliser ces informations et de donner un numéro IP à toute nouvelle machine qui boote. C'est comme cela que la machine « sait » dans quel sous-réseau elle se trouve et s'y intègre avec un numéro IP cohérent.

5.2 Les principaux services réseau

Il y a en fait plusieurs processus démons (plusieurs *services*) qui surveillent la carte réseau, selon le type de données que les paquets transportent. Cela permet de mettre en œuvre divers *protocoles de communication.* On peut citer : `ftp` (file transfert protocol, pour l'échange de fichiers), `smtp` (send mail transfert protocol, pour l'échange de courriers électroniques), `http` (hypertext transfert protocol, pour l'échange de pages web), `https` (un `http` sécurisé en cryptant les données, de sorte qu'un pirate a beaucoup plus de mal à décrypter les informations), `cups` (common unix printing system, communication avec les imprimantes), `ntp` (net time protocol, pour mettre une machine à l'heure), *etc.*

Le démon `sshd`, s'il tourne, permet à des utilisateurs extérieurs à la machine de se connecter sur la machine locale

3. Exception : dans certains cas, un paquet peutr être un « broadcast », ce qui signifie qu'il est envoyé à toutes les machines présentes physiquement sur le même (sous-)réseau.

4. Si certaines adresses sont utilisées vraiment très souvent, `root` a intérêt à les mémoriser localement sur la machine, dans le fichier `/etc/hosts`.

et d'y effectuer des commandes comme s'ils était physiquement au clavier local. Les utilisateur extérieurs peuvent utiliser `slogin user@nomMachine` pour se connecter, ou `ssh user@nomMachine commande` pour lancer une commande à distance, ou encore `scp fichierLocal user@nomMachine:adresse` pour copier des fichiers entre machines avec la même syntaxe que `cp`.

Pour des serveurs plus professionnels, on ouvrira aussi typiquement les services `ftpd` permettant aux utilisateurs de transférer des fichiers par exemple avec `lftp`, et Apache (démon `httpd`) pour constituer un serveur web, ainsi que `mysql` pour les bases de données.

Pour tous ces services, il ne faut pas confondre les machines qui peuvent les utiliser (appelées « clients ») et celles qui fournissent le service (appelées « serveurs »). Par exemple le fait de pouvoir lancer un `ssh` signifie qu'on demande à devenir client d'un démon `sshd` sur la machine distante, et que la machine distante est donc un serveur pour `ssh` ; *a priori* ce n'est pas symétrique. Il faut bien comprendre cela pour éviter d'offrir des services inutiles sur de nombreuses machines qui peuvent se contenter d'être client de chaque service. Chaque service fourni en tant que *serveur* est un trou de sécurité potentiel, qui doit donc être surveillé par le firewall. Il faut reconfigurer le firewall à chaque fois que l'on ouvre un nouveau service réseau pour qu'il le surveille.

5.3 Les protections nécessaires

Les distributions fournissent en général une paramétrisation relativement raisonnable de `sshd` du point de vue sécuritaire. Il est bon cependant que `root` vérifie le fichier de paramétrisation (typiquement `/etc/ssh/sshd_config`) s'il souhaite ouvrir ce service à ses utilisateurs sur certains serveurs.

Dans tous les cas, l'ouverture de l'un de ces services impose de mettre en place un firewall (pare-feu) comme `shorewall` pour surveiller les communications qui y passent et stopper les tentatives de piratage. La paramétrisation se fait le plus souvent par interface graphique.

Être serveur d'un service réseau quel qu'il soit impose aussi de mettre en place un démon, comme `msec` par exemple, qui vérifie quotidiennement que les droits des fichiers et surtout les « passe-droits » aux utilisateurs ne sont pas modifiés de manière malveillante. Il faut alors régulièrement lire les journaux du système (là encore, il y a des interfaces graphiques pour le faire) afin de vérifier si `msec` ou le firewall a signalé des anomalies. `msec` relève par exemple les fichiers ayant un « bit spécial » qui donne les droits du propriétaire du fichier à tout utilisateur qui exécute ce fichier, les fichiers en écriture non restreinte, le contrôle des `$PATH` abusives, *etc.*

Enfin, même les machines qui ne sont que clientes d'internet sont fragiles et nécessitent une surveillance « standard » par `msec` car les utilisateurs autorisent généralement les « scripts » des pages web qu'ils visitent. Ce sont des exécutables (java ou autres) qui peuvent être lancés sur la machine locale, chargés depuis la page web consultée et qui renvoient sur internet leurs résultats. Ils ont les droits des utilisateurs qui ont lancé le processus navigateur web et ils peuvent être malveillants...

Cela va sans dire : ne *jamais* consulter le web en tant que `root` !

6 Sauvegardes

Lorsqu'on gère une machine (même simplement son propre portable), il est ***crucial*** de faire des *sauvegardes régulières.* La fréquence des sauvegardes doit être guidée par la question simple « *quelle durée maximale de travail les utilisateurs acceptent-t-ils de perdre ?* ».

Il faut de plus distinguer deux fonctions différentes des sauvegardes : d'une part, les sauvegardes *de sûreté* pour ne pas perdre ses données en cas de crash-disk ou de piratage, d'autres part les sauvegardes préservant un historique pour retrouver des anciens fichiers supprimés (volontairement ou par inadvertance). Ces dernières nécessitent de définir une politique de sauvegarde plus élaborée.

Note : il est inutile de sauvegarder les fichiers du système, une réinstallation est rapide ; ce sont les fichiers impliquant du travail ou difficiles à retrouver ailleurs qui doivent être sauvegardés.

La sûreté des données est toujours assurée par la *redondance* des données. Il faut donc plusieurs copies des données qui changent régulièrement, et il faut que ces copies ne soient pas soumises aux même risques. En particulier, il ne faut pas qu'elles soient détruites simultanément en cas de feu ou autre catastrophe (donc diversifier les bâtiments de stockage).

Les logiciels de sauvegarde les plus courants sont `dump` (bien adapté pour les sauvegardes d'historiques), `tar` (transport

facile d'archives) et `rsync` (mises à jour de copies d'arborescences), et il est souvent plus simple de faire plusieurs copies sur des disques durs amovibles. Il faut cependant continuer à assurer la confidentialité des données, donc crypter toutes les sauvegardes sensibles et bien gérer les droits d'accès...

Pour des petits volumes de sauvegardes, on peut « griller » des DVD ou des Blu-rays, par exemple avec `k3b` ou tout autre graveur. Pour des sauvegarde de taille moyenne, on préférera des disques externes en achetant des NAS (Network Attached Storage) qui sont des serveurs de fichiers préconfigurés et pré-sécurisés que l'on branche directement sur le réseau. Enfin pour de gros volumes on achète des stations de sauvegarde dont le volume et les fréquences de sauvegardes globales (*via* le réseau) sont calibrés en fonction des besoins. Voir la commande `rsync`.

Enfin, il est beaucoup plus pratique de partitionner un système de telle sorte que les fréquences de sauvegarde de toutes les données d'une partition sont identiques. C'est même quasiment la seule motivation du partitionnement, outre d'éviter qu'un processus erroné puisse saturer tout le système de fichier du système.

6.1 Quelques commandes indispensables

Les commandes Unix suivantes sont très souvent utilisées (se reporter à l'annexe) :

> `cat`, `mkdir`, `rmdir`, `man`, `which`, `whoami`, `file`, `ls`, `ps et top`, `emacs`, `xterm`, `grep`, `more et less`, `sed`, `find`, `ooffice` (openoffice.org), `urpmf` ou similaire, `alias`, `gimp`, `lpr lpq et cancel`, `chmod`, `chown`, `chgrp`, `latex`, `rm`, `mv`, `tar`, `make`, `evince ou epdfview`, `thunderbird`, `firefox`...

De plus `find` et `tar` seront abordés ultérieurement.