The Unix and Internet Fundamentals HOWTO

by Eric S. Raymond v1.1, 3 Décembre 1998

Ce document décrit les principes fondamentaux des ordinateurs de type PC, des systèmes d'exploitation de type UNIX et d'Internet dans un langage non technique. (Traduction française Philippe Malinge *pmal@easynet.fr*)

Contents

1 Introduction

1.1 Sujet de ce document

Ce document est conçu pour aider les utilisateurs de Linux et d'Internet qui désirent apprendre en faisant. Bien que ce soit un bon moyen d'acquérir des compétences, quelquefois cela laisse de singulières lacunes dans la connaissance des bases – lacunes qui peuvent rendre difficile la réflexion créative ou perturber fortement, par manque d'un modèle mental clair sur ce qu'il devrait se passer.

J'essaierai de décrire clairement, dans un langage simple, comment tout marche. La présentation sera adaptée aux personnes qui utilisent Unix ou Linux sur du matériel de type PC. Cependant, je ferai ici couramment référence à 'Unix' : ce que je décrirai se retrouvera sur toutes les plates-formes et sur toutes les variantes d'Unix.

Je suppose que vous utilisez un PC avec un processeur de type Intel. Les détails diffèrent quelque peu si vous utilisez un processeur Alpha ou PowerPC ou une autre machine Unix, mais les concepts de base restent les mêmes.

Je ne voudrais pas répéter les choses, alors vous allez devoir faire attention, mais cela veut dire que vous retiendrez chaque mot que vous lirez. C'est une bonne idée que de tout parcourir rapidement la première fois ; vous devrez y revenir et relire un certain nombre de fois afin de digérer ce que vous avez appris.

C'est un document en permanente évolution. Je prévois d'ajouter des chapitres en réponse aux feedbacks, ainsi vous pourrez périodiquement le passer en revue.

1.2 Ressources rattachées

Si vous lisez dans l'espoir d'apprendre comment 'hacker', vous devrez lire *How To Become A Hacker FAQ* `http://www.tuxedo.org/~esr/faqs/hacker-howto.html` . Il y a beaucoup de liens vers d'autres ressources utiles.

1.3 Nouvelles versions de ce document

Les nouvelles versions de 'Unix and Internet Fundamentals HOWTO' seront postées périodiquement dans *comp.os.linux.help* et et *news.answers* `news:answers` . Elles pourront être téléchargées à partir de divers sites Linux WWW ou FTP, y compris la page d'accueil du LDP.

Vous pouvez accéder à la dernière version (en anglais) de ce document sur le World Wide Web via l'URL `http://sunsite.unc.edu/LDP/HOWTO/Unix-Internet-Fundamentals-HOWTO.html` .

1.4 Réactions et corrections

Si vous avez des questions ou des commentaires à propos de ce document, vous pouvez envoyer vos courriers électroniques à Eric S. Raymond, à *esr@thyrsus.com* . Toutes suggestions ou critiques seront les bienvenues. Seront spécialement appréciés les liens hypertexte vers des explications plus détaillées ou vers des concepts propres. Si vous trouvez des erreurs dans ce document, faites-le moi savoir afin que je puisse les corriger dans la nouvelle version. Merci.

2 Anatomie de base de votre ordinateur

Votre ordinateur possède un processeur à l'intérieur duquel se font réellement les calculs. Il possède une mémoire interne (ce que les gens DOS/Windows désignent par “RAM” et que les gens UNIX désignent souvent par “core”). Le processeur et la mémoire résident sur la *carte mère* qui est le coeur de votre ordinateur.

Votre ordinateur possède un écran et un clavier. Il a un (ou des) disque(s) dur(s) et un lecteur de disquettes. L'écran et vos disques ont des *cartes contrôleur* que l'on connecte sur la carte mère et qui aident l'ordinateur à piloter ces périphériques externes. Votre clavier est trop simple pour nécessiter une carte séparée ; le contrôleur est intégré dans le châssis du clavier.)

Nous décrirons plus tard en détails comment fonctionnent ces périphériques. Pour l'instant, quelques notions de base afin de garder à l'esprit comment ils fonctionnent ensemble :

Tous les éléments internes de votre ordinateur sont connectés par un *bus*. Physiquement, le bus est ce sur quoi vous connectez vos cartes contrôleur (carte vidéo, contrôleur disque, carte son si vous en avez une). Le bus est l'autoroute empruntée par les données entre votre processeur, votre écran, votre disque et le reste.

Le processeur, qui fait tout marcher, ne peut réellement voir tous les éléments directement ; il doit communiquer avec eux via le bus, le seul sous-système qui soit effectivement très rapide, qui accède directement à la mémoire (le core). Afin que les programmes puissent s'exécuter, ils doivent être en mémoire *core*.

Lorsque votre ordinateur lit un programme ou une donnée sur le disque, il se passe réellement les choses suivantes : le processeur utilise le bus pour envoyer une requête de lecture du disque à votre contrôleur de disque. Quelques instants après, le contrôleur de disque utilise le bus pour signaler à l'ordinateur qu'il a lu la donnée et qu'il l'a mise à un certain endroit de la mémoire. Le processeur peut utiliser le bus pour aller chercher ce qu'il y a à cet endroit de la mémoire.

Votre clavier et votre écran communiquent également avec le processeur via le bus mais d'une manière plus simple. Nous exposerons cela plus loin. Pour l'instant vous en savez suffisamment pour comprendre ce qu'il se passe lorsque vous allumez votre ordinateur.

3 Que se passe-t-il lorsque vous allumez votre ordinateur ?

Un ordinateur sans programme qui s'exécute est juste un tas inerte d'électronique. La première chose que doit faire un ordinateur lorsqu'il est allumé est de démarrer un programme spécial appelé *système d'exploitation*. Le travail du système d'exploitation est d'aider les autres programmes de l'ordinateur à travailler, en traitant les détails méprisables du contrôle du matériel de l'ordinateur.

Le processus de démarrage du système d'exploitation est appelé *booting* (originalement c'était *bootstrapping (laçage des chaussures)*, allusion à la difficulté d'enfiler soi même ses chaussures ‘par les lacets’. Votre ordinateur sait comment booter car les instructions de boot sont stockées dans un de ses composants, le composant BIOS (ou Basic Input/Output System).

Le composant BIOS dit où aller chercher, à une place fixe sur le disque dur de plus basse adresse (le *disque de boot*), un programme spécial appelé *chargeur de boot (boot loader)* (sous Linux le chargeur de boot est appelé LILO). Le chargeur de boot est chargé en mémoire puis lancé. Le travail du chargeur de boot est de démarrer le système d'exploitation réel.

Le chargeur fait cela en allant chercher un *noyau*, en le chargeant en mémoire et en le démarrant. Lorsque vous bootez Linux et voyez "LILO" sur l'écran suivi par une succession de points, c'est qu'il charge le noyau. (Chaque point signifie qu'il vient de charger un autre *bloc du disque* du code du noyau.)

(Vous pouvez vous demander pourquoi le BIOS ne charge pas le noyau directement – pourquoi ces deux étapes du processus avec le chargeur de boot ? C'est que le BIOS n'est pas vraiment intelligent. En fait il est carrément stupide, et Linux ne l'utilise jamais après avoir booté. A l'origine, j'ai programmé sur des PC 8-bits primitifs avec de petits disques : littéralement ils ne pouvaient accéder à suffisamment de disque pour charger le noyau directement. L'étape du chargeur de boot vous permet de démarrer plusieurs systèmes d'exploitation à partir de différents emplacements de votre disque, dans le cas où Unix n'est pas assez bon pour vous.)

Une fois que le noyau démarre, il doit chercher autour de lui, trouver le reste du matériel et être prêt pour exécuter des programmes. Il fait cela non pas en fouillant à des adresses mémoire ordinaires, mais à des *ports d'Entrée/Sortie* – des adresses spéciales du bus, sensées avoir une carte contrôleur de périphériques en attente de commandes à cet endroit. Le noyau ne fouille pas au hasard ; il a un ensemble de connaissances qui lui permet de savoir ce qu'il est sensé trouver ici, et comment les contrôleurs répondraient s'ils étaient présents. Ce processus est appelé *Exploration automatique*.

La plupart des messages que vous voyez au moment du boot sont l'exploration de votre matériel par le noyau à travers les ports d'Entrée/Sortie, le chiffrage de ce qui est disponible et l'adaptation à votre machine. Le noyau Linux est extrêmement bon pour cela, meilleur que la plupart des autres Unix et *tellement* meilleur que DOS ou Windows. En fait, beaucoup de vieux adeptes de Linux pensent que l'ingéniosité des explorations de Linux lors du boot (qui lui permettent de s'installer relativement simplement) ont été une raison de s'épanouir dans le monde des expériences des Unix libres pour attirer une masse critique d'utilisateurs.

Mais rendre le noyau complètement chargé et s'exécutant n'est pas la fin du processus de boot ; c'est juste la première étape (quelquefois appelée *niveau d'exécution 1 (run level 1)*).

L'étape suivante du noyau est de s'assurer que vos disques sont OK. Les systèmes de fichiers sur disques sont des choses fragiles ; s'ils ont été endommagés par une panne matérielle ou par une coupure soudaine d'alimentation électrique, il y a de bonnes raisons de rétablir l'intégrité avant que votre Unix ne puisse aller plus loin. Nous parlerons plus tard de ce que l'on dit à propos de 8.5 (comment les systèmes de fichiers peuvent devenir mauvais).

L'étape suivante du noyau est de lancer plusieurs *démons*. Un démon est un programme comme un spouleur d'imprimante, un serveur de mail ou un serveur WWW qui se cache en arrière-plan en attendant d'avoir des choses à faire. Ces programmes spéciaux doivent coordonner plusieurs requêtes qui peuvent entrer en conflit. Il y a des démons car il est souvent plus facile d'écrire un programme qui s'exécute constamment et qui sait tout des requêtes, plutôt que d'essayer de s'assurer qu'un troupeau de copies (chacune traitant une requête et toutes s'exécutant en même temps) ne se gêneraient pas mutuellement. La collection particulière de démons que le système démarre peut varier, mais inclura presque toujours un spouleur d'imprimante (un démon garde-barrière de votre imprimante).

Une fois que tous les démons ont démarré, nous sommes dans le *niveau d'exécution 2 (run level 2)*. L'étape suivante est la préparation pour les utilisateurs. Le noyau démarre une copie d'un programme appelé `getty` pour surveiller votre console (et peut être d'autres copies pour surveiller des ports-série entrants) Ce programme est celui duquel jaillit le prompt `login` sur votre console. Nous sommes maintenant dans le *niveau d'exécution 3 (run level 3)* et prêts pour votre connexion et l'exécution de vos programmes.

Quand vous vous connectez (en donnant un nom et un mot de passe), vous vous identifiez auprès de `getty` et de l'ordinateur. Il exécute maintenant un programme appelé (assez naturellement) `login`, qui réalise des tâches ancillaires et démarre un interpréteur de commandes, le *shell*. (Oui `getty` et `login` pourraient être un seul et même programme. Ils sont séparés pour des raisons historiques que nous n'expliciterons pas ici.)

Dans la section suivante, nous parlerons de ce qui se passe lorsque vous exécutez des programmes à partir du shell.

4 Que se passe-t-il lorsque vous exécutez des programmes à partir du shell?

Le shell normal vous donne le prompt '$' que vous voyez après vous être connecté (cependant vous pouvez le modifier et mettre autre chose). Nous ne parlerons pas de la syntaxe du shell et des choses faciles que vous pouvez voir sur votre écran ici ; alors que nous 'jetterons un oeil' sur ce qu'il se passe du point de vue de l'ordinateur.

Après la phase de boot et avant que vous n'exécutiez un programme, vous pouvez penser à votre ordinateur comme étant un zoo de processus qui attendent qu'il se passe quelque chose. Ils attendent des *événements*. Un événement, ce peut être l'enfoncement d'une touche ou un déplacement de la souris. Ou, si votre machine est connectée à un réseau, un événement peut être un paquet de données venant de ce réseau.

Le noyau est un de ces processus. C'en est un spécial, car il contrôle le moment où les autres processus *utilisateur* peuvent s'exécuter, et c'est normalement le seul processus qui accède directement au matériel de la machine. En fait, les processus utilisateurs font des requêtes au noyau lorsqu'ils veulent obtenir une entrée clavier, écrire sur votre écran, lire ou écrire sur votre disque ou juste autre chose que consommer quelques bits en mémoire. Ces requêtes sont appelées *appels système*.

Normalement toute Entrée/Sortie passe par le noyau de manière à ce qu'il puisse ordonnancer les opérations et éviter ainsi aux processus de se marcher les uns sur les autres. Quelques processus utilisateur sont autorisés à contourner le noyau, habituellement en ayant accès directement aux ports d'Entrée/Sortie. Les serveurs X (les programmes qui traitent les requêtes graphiques des autres programmes sur la plupart des machines Unix) sont des exemples classiques. Mais nous n'avons pas vu de serveur X pour l'instant ; vous êtes au prompt du shell sur une console en mode caractères.

Le shell est juste un processus utilisateur, et non un processus particulièrement spécial. Il attend vos frappes sur les touches du clavier, écoutant (à travers le noyau) le port d'E/S du clavier. Comme le noyau les voit, il les affiche sur votre écran et les passe au shell. Le shell essaie de les interpréter comme étant des commandes.

Tapez 'ls' suivi de 'Enter' afin de lister le contenu d'un répertoire. Le shell applique ses règles internes pour évaluer la commande que vous voulez exécuter dans le fichier '/bin/ls'. Il fait un appel système en demandant au noyau de lancer '/bin/ls' comme un processus *fils* et donne son accès à l'écran et au clavier à travers le noyau. Le shell se rendort en attendant que 'ls' se termine.

Lorsque /bin/ls est terminé, il dit au noyau qu'il a terminé en effectuant un appel système *exit*. Le noyau réveille le shell et lui dit qu'il peut continuer à s'exécuter. Le shell affiche un autre prompt et attend une autre ligne en entrée.

D'autres choses peuvent être faites pendant l'exécution de 'ls', cependant (nous supposerons que la liste du répertoire est très longue). Vous pourriez basculer sur une autre console virtuelle, vous connecter, et lancer une jeu de Quake par exemple. Ou bien, supposez que vous êtes connecté à Internet : votre machine peut envoyer ou recevoir des mails pendant que '/bin/ls' s'exécute.

5 Comment marchent les périphériques d'entrée et les interruptions ?

Votre clavier est un périphérique très simple ; simple car il génère un petit flux de données très lentement (sur un ordinateur standard). Lorsque vous relâchez une touche, cet événement est signalé par le câble du clavier qui va provoquer une *interruption matériel*.

C'est au système d'exploitation de surveiller de telles interruptions. Pour chaque type possible d'interruption, il y a un *handler d'interruption*, une partie du système d'exploitation dissimule toutes les données associées (comme la valeur touche enfoncée/touche relâchée) tant qu'elle ne peut être traitée.

Ce que le fait le handler d'interruption disque pour votre clavier est de déposer la valeur de la touche dans une zone en bas de la mémoire (core). Ainsi elle sera disponible pour l'inspection lorsque le système d'exploitation passera le contrôle à n'importe quel programme supposé attendre présentement une entrée clavier.

Des périphériques d'entrée plus complexes comme les disques travaillent de manière similaire. Précédemment nous faisions référence à un contrôleur de disques utilisant le bus pour signaler qu'une requête disque a bien été exécutée. Que se passe-t-il si ce disque reçoit une interruption ? Le handler de l'interruption disque copie alors la donnée trouvée dans la mémoire, pour une utilisation future par le programme qui en avait fait la demande.

Chaque type d'interruption est associé à un *niveau de priorité*. Les interruptions de plus basse priorité (comme les évènements clavier) sont traitées après celles de priorité supérieures (comme les tops d'horloge ou les événements disque). Unix a été conçu pour traiter prioritairement les types d'événements qui doivent être traités rapidement afin de conserver une machine sur laquelle les temps de réponse sont sont sans à-coup.

Les messages que vous voyez pendant la phase de boot font référence à des numéros d'*IRQ*. Vous devez être prévenus qu'une des causes les plus courantes de mauvaise configuration de votre matériel est d'avoir deux périphériques qui essaient d'utiliser la même IRQ, sans savoir ce que c'est réellement.

La réponse est ici. IRQ est l'abbréviation de "Interrupt ReQuest". Le système d'exploitation a besoin de savoir au démarrage quel numéro d'interruption sera utilisé par chaque périphérique, ainsi il peut associer le handler adéquat pour chacun. Si deux périphériques différents essaient d'utiliser la même IRQ, les interruptions seraient quelquefois distribuées au mauvais handler. Cela est classique au moins au verrouillage du périphérique, et peut parfois déstabiliser le système d'exploitation, qu'il se "désintègre" ou qu'il se crashe.

6 Comment mon ordinateur fait-il plusieurs choses en même temps ?

En fait, il ne le fait pas. Les ordinateurs ne peuvent traiter qu'une seule tâche (ou *processus*) à la fois. Mais un ordinateur peut changer de tâche très rapidement, et duper l'esprit humain en lui faisant croire qu'il fait plusieurs choses en même temps. C'est ce que l'on appelle le *temps partagé*.

Une des tâches du noyau est de gérer le temps partagé. C'est une partie dédiée à l'*ordonnanceur* qui conserve chez lui toutes les informations sur les autres processus (non noyau) de votre environnement. Chaque 1/60 ème de seconde, une horloge avertit le noyau, générant une interruption horloge. L'ordonnanceur arrête le processus qui s'exécute, le suspend dans l'état, et donne le contrôle à un autre processus.

1/60 ème de seconde peut paraître peu de temps. Mais sur les microprocesseurs actuels c'est assez pour exécuter des dizaines de milliers d'instructions machine, ce qui permet d'effectuer beaucoup de choses. Même si vous avez plusieurs processus, chacun peut accomplir un petit peu sa tâche pendant ses tranches de temps.

En pratique, un programme ne dispose pas de sa tranche de temps entière. Si une interruption arrive d'un périphérique d'E/S, le noyau arrêtera en réalité la tâche courante, exécutera le handler d'interruption et retournera à la tâche courante. Une tempête d'interruption de haute priorité peut interdire tout traitement normal ; ce mauvais comportement est appelé *défaite (thrashing)* et est difficile à provoquer sur les Unix modernes.

En fait, la vitesse des programmes est très rarement limitée par le temps machine qu'ils peuvent obtenir (il y a quelques exceptions à cette règle, comme la génération de son ou de graphiques en 3-D. Le plus souvent, les délais sont dus à l'attente, par le programme, des données d'un disque ou d'une connexion réseau.

Un système d'exploitation qui peut supporter de manière routinière plusieurs processus est appelé "multitâche". Les systèmes d'exploitation de la famille Unix ont été conçus dès le début pour le multitâche et sont vraiment bons pour ça – beaucoup plus efficaces que celui de Windows et MAC OS, pour lesquels le multitâche a été introduit a posteriori et qui le traitent plutôt pauvrement. Efficace, multitâche, fiable sont quelques-unes des raisons qui rendent Linux supérieur pour le réseau, les communications et les services WEB.

7 Comment mon ordinateur évite aux processus d'empiéter les uns sur les autres ?

L'ordonnanceur du noyau fait attention à séparer les processus dans le temps. Votre système d'exploitation les divise aussi dans l'espace, de telle manière que ces processus n'empiètent pas sur la mémoire de travail des autres. Ces choses que votre système d'exploitation réalise sont appelées *gestion de la mémoire*.

Chaque processus de votre 'troupeau' a besoin de son propre espace mémoire afin de mettre son code et de garder des variables et leur résultat. Vous pouvez imaginer cet ensemble constitué d'un *segment de code* accessible en lecture uniquement (contenant les instrucions du processus) et un *segment de données* accessible en écriture (contenant toutes les variables du processus). Le segment de données est véritablement propre à chaque processus, mais si deux processus exécutent le même code, Unix s'arrange automatiquement pour qu'ils partagent le même segment de code dans un soucis d'efficacité.

L'efficacité est importante car la mémoire est chère. Quelquefois, vous ne disposez pas de suffisamment de mémoire pour faire tenir tous les programmes, spécialement si vous utilisez un gros programme comme un serveur X-WINDOW. Pour contourner cela, Unix utilise une stratégie appelée *mémoire virtuelle*. Cela n'essaie pas de faire tenir tout le code et les données d'un processus en mémoire. Cependant, il garde seulement un espace de travail ; le reste de l'état du processus est laissé dans un endroit spécial sur votre disque : *l'espace d'échange (swap space)*.

Lorsque le processus s'exécute, Unix essaie d'anticiper comment l' espace de travail changera, et ne chargera en mémoire que les morceaux dont il a besoin. Faire cela efficacement est compliqué et délicat, je n'essaierai pas de le décrire ici – mais cela dépend du fait que le code et les références aux données peuvent arriver en blocs, avec chaque nouveau référençant vraisemblablement un proche ou un ancien. Ainsi, si Unix garde le code ou les données fréquemment (ou récemment) utilisés, vous gagnerez du temps.

Notez que dans le passé, le "quelquefois" que nous employons deux paragraphes plus haut était "souvent" voire "toujours", – la taille de la mémoire était habituellement petite par rapport à la taille des programmes en cours d'exécution, de telle manière que les échanges entre le disque et la mémoire ("swapping") étaient fréquents. La mémoire est beaucoup moins chère de nos jours et même les machines bas de gamme en sont bien dotées. Sur les machines mono-utilisateur avec 64Mo de mémoire, il est possible de faire tourner X-WINDOW et un mélange de programmes sans jamais swapper.

Même dans cette situation joyeuse, la part du système d'exploitation appelée le *gestionnaire de mémoire* a un important travail à faire. Il doit être sûr que les programmes ne peuvent modifier que leurs segments de mémoire – ce qui empêche un code erroné ou malicieux dans un programme de ramasser les données dans un autre. Pour faire cela, il conserve une table des segments de données et de code. La table est mise à jour chaque fois qu'un processus demande de la mémoire ou en libère (habituellement plus tard lorsqu'il se termine).

Cette table est utilisée pour passer des commandes à une partie spécialisée du matériel sous-jacent appelée un *UGM (MMU)* ou *unité de gestion mémoire (memory management unit)*. Les processeurs modernes disposent de MMUs intégrés. Le MMU a la faculté de mettre des barrières autour de zones mémoire, ainsi une référence en "dehors des clous" sera refusée et générera une interruption spéciale pour être traitée.

Si vous avez déjà vu le message Unix qui dit "Segmentation fault", "core dumped" ou quelque chose de similaire, c'est exactement ce qu'il se passe ; un programme en cours d'exécution a tenté d'accéder à de la mémoire en dehors de son segment et a provoqué une interruption fatale. Cela indique un bug dans le code du programme ; le *core dump* laisse une information en vue d'un diagnostic à l'attention du programmeur afin qu'il puisse trouver la trace de son erreur.

8 Comment mon ordinateur stocke des choses sur le disque ?

Sur votre disque dur sous Unix, vous voyez un arbre de répertoires nommés et des fichiers. Normalement vous ne devriez pas à chercher à en savoir plus, mais cela peut s'avérer utile de savoir ce qu'il y a dessous si vous avez un crash disque et besoin d'essayer de nettoyer des fichiers. Malheureusement il n'y a pas de bon moyen de décrire l'organisation du disque en partant du niveau fichier et en descendant, c'est pour cela que je le décrirai en remontant à partir du niveau matériel.

8.1 Bas niveau du disque et structure du système de fichiers

La surface de votre disque , sur laquelle il stocke les données est divisée comme une cible de jeu de fléchettes – en pistes circulaires qui sont partagées en secteurs. Parce que les pistes de l'extérieur contiennent plus de surface que celles près de l'axe de rotation, au centre du disque, les pistes externes ont plus de secteurs que celles de l'intérieur. Chaque secteur (ou *bloc disque*) a la même taille, qui est généralement de 1Ko (1024 mots de 8 bits). Chaque bloc disque a une adresse unique ou un *numéro de bloc disque*.

Unix divise le disque en *partitions disque*. Chaque partition est une succession de blocs qui est utilisée indépendamment des autres partitions, comme un système de fichiers ou un espace d'échange (swap space). La partition ayant le plus petit numéro est souvent traitée spécialement, telle la *partition de boot* dans laquelle vous pouvez mettre un noyau pour booter.

Chaque partition est soit un *espace de swap* (utilisé pour implémenter la 7 (mémoire virtuelle)) soit un *système de fichiers* pour stocker des fichiers. Les partitions de swap sont traitées comme une séquence linéaire de blocs. Les systèmes de fichiers d'un autre coté, ont besoin de relier les noms de fichiers à des séquences de blocs disque. Parce que les fichiers grossissent, diminuent, et changent tout le temps, les blocs de données d'un fichier ne seront pas une séquence linéaire mais pourront être dispersés sur toute la partition (tant que le système d'exploitation pourra trouver un bloc libre).

8.2 Noms de fichiers et répertoires

Dans chaque système de fichiers, la liaison entre les noms et les blocs est réalisée grâce à une structure appelée *i-node (noeud d'index)*. Il y en a tout un tas proche de la "base" (numéro de bloc les plus faibles) du système de fichiers (les tout premiers sont utilisés pour des besoins d'intégrité et de label que nous ne décrirons pas ici). Chaque i-node décrit un fichier. Les blocs de données des fichiers sont au dessus des i-nodes (conceptuellement).

Chaque i-node contient la liste des numéros des blocs du fichier (réellement c'est une demi-vérité, c'est seulement valable pour les petits fichiers, mais le reste de ces détails ne sont pas importants ici). Notez que l'i-node *ne contient pas* le nom du fichier.

Les noms des fichiers résident dans les *structures de répertoires*. Une structure de répertoire contient juste une table des noms et des numéros d'i-node associés. C'est la raison pour laquelle, sous Unix, un fichier peut avoir plusieurs noms réels (ou *liens forts (hard links)*) ; Il y a juste plusieurs entrées dans un répertoire qui pointent vers le même i-node.

8.3 Points de montage

Dans le cas le plus simple, votre système de fichiers Unix tient sur une seule partition disque. Cependant vous verrez que cette disposition sur des petits systèmes Unix n'est pas pratique. Typiquement il est réparti sur plusieurs partitions disque voire sur plusieurs disques physiques. Ainsi par exemple, votre système peut avoir une petite partition où le noyau réside, une un peu plus grande pour les utilitaires du système et une beaucoup plus grosse pour les répertoires des utilisateurs.

La seule partition à laquelle vous aurez accès immédiatement après le boot est votre *partition racine (root partition)*, qui est (presque toujours) celle à partir de laquelle vous avez booté. Elle contient le répertoire racine du système de fichiers, le noeud le plus haut à partir duquel tout est raccroché.

Les autres partitions du système doivent être attachées à cette racine afin que votre système de fichiers unique ou multi-partition soit accessible. Au milieu du processus de boot, votre Unix rendra ces partitions 'non root' accessibles. Il devra *monter* chacune d'elles sur un répertoire de la partition racine.

Par exemple, si votre Unix a un répertoire appelé '/usr', c'est probablement un point de montage d'une partition qui contient un tas de programmes installés avec votre Unix mais qui ne sont pas nécessaires durant la phase initiale de boot.

8.4 Comment un fichier est retrouvé ?

Maintenant nous pouvons considérer le système de fichiers dans une démarche descendante. Lorsque vous ouvrez un fichier (tel que `/home/esr/WWW/ldp/fundamentals.sgml`) voici ce qu'il arrive :

Votre noyau démarre de la racine de votre système de fichiers Unix (dans la partition root). Il cherche un répertoire appelé 'home'. Habituellement 'home' est un point de montage d'une grande partition pour les utilisateurs, il descend à l'intérieur. Au sommet de la structure du répertoire de cette partition utilisateur, il va chercher une entrée nommée 'esr' et en extraire le numéro d'i-node. Il ira à cette i-node, notez que c'est une structure de répertoire, et retrouvera 'WWW'. En exploitant *cet* i-node, il ira au sous répertoire correspondant et retrouvera 'ldp'. Ce qui lui donnera encore un autre i-node répertoire. En ouvrant ce dernier, il trouvera un numéro d'i-node pour 'fundamentals.sgml'. Cet i-node n'est pas un répertoire mais fournit la liste des blocs associés au fichier.

8.5 Comment les choses peuvent dégénérer ?

Plus haut, nous avons laissé entendre que les systèmes de fichiers étaient fragiles. Maintenant nous savons que pour accéder à un fichier vous devez parcourir une longue chaîne arbitraire de références à des répertoires et à des inodes. A présent, supposons que votre disque dur possède une zone défectueuse.

Si vous êtes chanceux, il détruira quelques données d'un fichier. Si vous êtes malchanceux, il va corrompre une structure de répertoire ou un numéro d'inode et laissera un sous arbre entier de votre système dans l'oubli – ou, pire, cela a donné une structure corrompue qui pointe par plusieurs chemins au même bloc disque ou inode. Une telle corruption peut s'étendre par des opérations courantes sur les fichiers qui ne se trouvent pas au point d'origine.

Heureusement, ce genre de d'imprévu devient de plus en plus rare car les disques sont de plus en plus fiables. Malgré tout, cela veut dire que votre Unix voudra vérifier périodiquement l'intégrité du système de fichiers afin de s'assurer que rien ne cloche. Les Unix modernes font une vérification rapide sur chaque partition au moment du boot, juste avant de les monter. Au bout d'un certain nombre de redémarrages (reboot), la vérification sera plus approfondie et durera quelques minutes.

Si tout cela vous parait, comme Unix, terriblement complexe et prédisposé aux défaillances, au contraire, c'est rassurant de savoir que ces vérifications faites au démarrage de la machine, détectent et corrigent les problèmes courants *avant* qu'ils ne deviennent réellement désastreux. D'autres systèmes d'exploitation ne disposent pas de ces fonctionnalités, qui accélèrent un petit peu le démarrage, mais peuvent vous laisser tout 'bousiller' en essayant de récupérer à la main (et en supposant que vous ayez une copie des Utilitaires Norton ou autre à portée de main...).

9 Comment fonctionnent les langages d'ordinateur ?

Nous avons déjà évoqué 4 (comment les programmes sont exécutés). Chaque programme en fin de compte doit exécuter une succession d'octets qui sont les instructions dans le *langage machine* de votre ordinateur. Les humains ne pratiquent pas très bien le langage machine ; cela est devenu rare, art obscur même parmi les hackers.

La plupart du code du noyau d'Unix excepté une petite partie de l'interface avec le matériel est de nos jours écrite dans un *langage de haut niveau*. (Le terme 'haut niveau' est un héritage du passé afin de le distinguer du 'bas-niveau' des *langages assembleur*, qui sont de maigres "couches" autour du code machine.

Il y plusieurs types différents de langages de haut niveau. Afin de parler d'eux, vous trouverez utile que j'attire votre attention sur le fait que le *code source* d'un programme (la création humaine, la version éditable) est passé à travers plusieurs types de traductions pour arriver en code machine, que la machine peut effectivement exécuter.

9.1 Langages compilés

Le type le plus classique de langage est un *langage compilé*. Les langages compilés sont traduits en fichiers exécutables de code machine binaire par un programme spécial appelé (assez logiquement) un *compilateur*. Lorsque le binaire est généré, vous pouvez l'exécuter directement sans regarder à nouveau dans le code source. (La plupart des logiciels délivrés sous forme de binaires compilés sont faits à partir d'un source auquel vous n'avez pas accès.)

Les langages compilés tendent à fournir une excellente performance et ont un accès le plus complet au système d'exploitation, mais il difficile de programmer avec.

Le langage C, langage dans lequel chaque Unix est lui-même écrit, est de tous le plus important (avec sa variante C++). FORTRAN est un autre langage compilé qui reste utilisé par de nombreux ingénieurs et scientifiques mais plus vieux et plus primitif. Dans le monde Unix aucun autre langage compilé n'est autant utilisé. En dehors de lui, COBOL est très largement utilisé pour les logiciels de finance et comptabilité.

Il y a bien d'autres compilateurs de langages, mais la plupart sont en voie d'extinction ou sont strictement des outils de recherche. Si vous êtes un nouveau développeur Unix qui utilise un langage compilé, il est incontournable que ce soit C ou C++.

9.2 Langages interprétés

Un *langage interprété* dépend d'un programme interpréteur qui lit le code source et traduit à la volée en calculs et appels système. Le source doit être ré-interprété (et l'interpréteur présent) à chaque fois que le programme est exécuté.

Les langages interprétés tendent à être plus lents que les langages compilés, et limitent souvent les accès au système d'exploitation ou au matériel sous-jacent. D'un autre côté, il est plus facile de programmer et ils tolèrent plus d'erreurs de codage que les langages compilés.

Quelques utilitaires Unix, incluant le shell et bc(1) et sed(1) et awk(1), sont effectivement des petits langages interprétés. Les BASICs sont généralement interprétés. Ainsi est Tcl. Historiquement, le langage le plus interprété était LISP (une amélioration énorme sur la plupart de ses successeurs). Aujourd'hui, Perl est très largement utilisé et devient résolument plus populaire.

9.3 Langages P-code

Depuis 1990 un type de langage hybride qui utilise la compilation et l'interprétation est devenu incroyablement important. Les langages P-code sont comme des langages compilés dans le sens où le code est traduit dans une forme binaire compacte qui est celle que vous exécutez, mais cette forme n'est pas du code machine. Au lieu de cela, c'est du *pseudo-code* (ou *p-code*), qui est généralement un peu plus simple mais plus puissant qu'un langage machine réel. Lorsque vous exécutez le programme, vous interprétez du p-code.

Le p-code peut s'exécuter pratiquement aussi rapidement que du binaire compilé (les interpréteurs de p-code peuvent être relativement simples, petits et rapides). Mais les langages p-code peuvent garder la flexibilité et la puissance d'un bon interpréteur.

D'importants langages p-code sont Python et Java.

10 Comment Internet fonctionne ?

Afin de vous aider à comprendre comment Internet fonctionne, nous verrons ce qui se passe lorsque vous effectuez une opération classique – pointer dans un navigateur ce document à partir du site Web de référence du Projet de Documentation de Linux (Linux Documentation Project). Ce document est :

`http://sunsite.unc.edu/LDP/HOWTO/Fundamentals.html`

ce qui veut dire qu'il réside dans le fichier LDP/HOWTO/Fundamentals.html, sous le répertoire exporté World Wide Web de la machine sunsite.unc.edu.

10.1 Noms et localisations

La première chose que votre navigateur doit faire est d'établir une connexion réseau avec la machine sur laquelle se trouve le document. Pour faire cela, il doit tout d'abord trouver la localisation réseau de *l'hôte* sunsite.unc.edu (hôte est un raccourci pour 'machine hôte' ou 'hôte réseau' ; sunsite.unc.edu est un *nom d'hôte (hostname)* typique). La localisation correspondante est en fait un nombre appelé *adresse IP* (nous expliquerons la partie 'IP' de ce terme plus tard).

Pour faire cela, votre navigateur sollicite un programme nommé *serveur de noms*. Le serveur de noms peut résider sur votre machine, mais il est plus probable qu'il soit sur une machine de service avec laquelle vous pouvez dialoguer. Lorsque vous abonnez chez un Fournisseur d'Accés à Internet (FAI), une partie de la procédure d'installation décrit certainement la manière d'indiquer à votre logiciel Internet l'adresse IP du serveur de noms du réseau du FAI.

Les serveurs de noms sur différentes machines communiquent avec les autres en échangeant et en gardant à jour toutes les informations nécessaires à la résolution de noms d'hôte (en les associant à des adresses IP). Votre serveur de noms doit demander à trois ou quatre sites à travers le réseau afin de résoudre sunsite.unc.edu, mais cela se déroule vraiment rapidement (en moins d'une seconde).

Le serveur de noms dira à votre navigateur que l'adresse IP de Sunsite est 152.2.22.81 ; sachant cela, votre machine sera capable d'échanger des bits avec Sunsite directement.

10.2 Paquets et routeurs

Ce que le navigateur veut faire est d'envoyer une commande au serveur Web sur Sunsite qui a la forme suivante :

```
GET /LDP/HOWTO/Fundamentals.html HTTP/1.0
```

Que se passe-t-il alors ? La commande est faite de *paquets* ; un bloc de bits comme un télégramme est découpé en trois choses importantes : *l'adresse source* (l'IP de votre machine), *l'adresse destination* (152.2.22.81), et le *numéro de service* ou *numéro de port* (80, dans ce cas) qui indique que c'est une requête World Wide Web.

Alors votre machine envoie le paquet par le fil (de la connexion modem avec votre FAI, ou le réseau local) jusqu'à ce qu'il rencontre une machine spécialisée appelée *routeur*. Le routeur possède une carte de l'Internet dans sa mémoire – pas une complète mais une qui décrit votre voisinage réseau et sait comment aller aux routeurs pour les autres voisinages sur l'Internet.

Votre paquet peut passer à travers plusieurs routeurs sur le chemin de sa destination. Les routeurs sont adroits. Ils regardent combien de temps prend un accusé réception pour recevoir un paquet. Ils utilisent cette information pour aiguiller le trafic sur les liens rapides. Ils l'utilisent pour s'apercevoir que d'autres routeurs (ou un câble) sont déconnectés du réseau et modifier le chemin si possible en trouvant une autre route.

Il existe une légende urbaine qui dit qu'Internet a été conçu pour survivre a une guerre nucléaire. Ce n'est pas vrai, mais la conception d'Internet est extrêmement bonne en ayant une performance fiable basé sur des couches matérielles d'un monde incertain... C'est directement du au fait que son intelligence est distribuée à travers des milliers de routeurs plutôt qu'à quelques auto-commutateurs massifs (comme le réseau téléphonique). Cela veut dire que les défaillances tendent à être bien localisées et le réseau peut les contourner.

Une fois que le paquet est arrivé à destination, la machine utilise le numéro de service pour le fournir au serveur Web. Le serveur Web peut savoir à qui répondre en regardant l'adresse source du paquet. Quand le serveur Web renvoie ce document, il sera coupé en plusieurs paquets. La taille des paquets varie en fonction du média de transmission du réseau et du type de service.

10.3 TCP et IP

Pour comprendre comment des transmissions de multiples paquets sont réalisées, vous devez savoir que l'Internet utilise actuellement deux protocoles empilés l'un sur l'autre.

Le plus bas niveau, *IP* (Internet Protocol), sait comment recevoir des paquets individuels d'une adresse source vers une adresse destination (c'est pourquoi elles sont appelées adresses IP). Cependant, IP n'est pas fiable ; si un paquet est perdu ou jeté, les machines source et destination ne le sauront jamais. Dans le jargon réseau, IP est un protocole *sans connexion (ou mode non connecté)* ; l'expéditeur envoie juste un paquet au destinataire et n'attend jamais un accusé de réception.

Cependant, IP est rapide et peu coûteux. Quelquefois, rapide, peu coûteux et non fiable c'est OK. Lorsque vous jouez en réseau à Doom ou Quake, chaque balle est représentée par un paquet IP. Si quelques-unes sont perdues, c'est OK.

Le niveau supérieur, *TCP* (Transmission Control Protocol), fournit la fiabilité. Quand deux machine négocient une connexion TCP (ce qu'elles font en utilisant IP), le destinataire doit envoyer des accusés de réception des paquets qu'il reçoit à l'expéditeur. Si l'expéditeur ne reçoit pas un accusé de réception pour un paquet après un certain temps, il renvoie ce paquet. De plus, l'expéditeur donne à chaque paquet TCP un numéro de séquence, que le destinataire peut utiliser pour ré-assembler les paquets dans le cas où il sont arrivés dans le désordre. (Cela peut arriver si les liens réseau se rétablissent ou cassent pendant une connexion.)

Les paquets TCP/IP contiennent également un checksum pour permettre la détection de données altérées par de mauvais liens. Ainsi, du point de vue de quelqu'un utilisant TCP/IP et des serveurs de noms, il ressemble à une voie fiable pour faire passer des flux d'octets entre des paires hôte/numéro de services. Les gens qui écrivent des protocoles réseau ne doivent pas se soucier la plupart du temps de la taille des paquets, du ré-assemblage des paquets, de la vérification d'erreurs, le calcul du checksum et la retransmission qui sont au niveau inférieurs.

10.4 HTTP, un protocole d'application

Maintenant revenons à notre exemple. Les navigateurs et les serveurs Web parlent un *protocole d'application* qui est au dessus de TCP/IP, en l'utilisant simplement comme une manière de passer des chaînes d'octets dans les deux sens. Ce protocole est appelé *HTTP* (Hyper-Text Transfer Protocol) et nous en avons déjà vu une commande – la commande GET utilisée ci-dessus.

Lorsque la commande GET arrive au serveur Web de sunsite.unc.edu avec comme numéro de service 80, elle sera expédiée à un *démon serveur* qui écoute le port 80. La plupart des services Internet sont implémentés par des démons serveurs qui ne font rien d'autre qu'attendre sur des numéros de port, récolter et exécuter les commandes entrantes.

Cette conception de l'Internet a une règle qui prime sur les autres, c'est que toutes les parties sont le plus simple possible et humainement accessible. HTTP, et ses compères (comme le Simple Mail Transfer Protocol, *SMTP*, qui est utilisé pour transporter du courrier électronique entre des machines) utilisent de simples commandes de texte qui se terminent par un retour chariot.

C'est rarement inefficace ; dans certaines circonstances vous pouvez obtenir plus de rapidité en employant un protocole binaire fortement codé. Mais l'expérience a montré que le bénéfice d'avoir des commandes qui sont faciles à décrire et à comprendre l'emportent sur le gain marginal de l'efficacité que l'on peut espérer au prix de choses compliquées et compactes.

Par conséquent, ce que le démon serveur vous renvoie via TCP/IP est aussi du texte. Le début de la réponse ressemblera à quelque chose comme (quelques en-têtes ont été supprimés) :

```
HTTP/1.1 200 OK
Date: Sat, 10 Oct 1998 18:43:35 GMT
Server: Apache/1.2.6 Red Hat
Last-Modified: Thu, 27 Aug 1998 17:55:15 GMT
Content-Length: 2982
Content-Type: text/html
```

Ces en-têtes seront suivis d'une ligne vide et du texte de la page Web (après que la connexion sera rompue). Votre navigateur affichera simplement cette page. Les en-têtes indiquent – en particulier, l'en-tête Type de Contenu (Content-Type) – comment les données reçues sont vraiment du HTML).