Tribune Libre

Adaptation française du livre
« Open Sources : Voices of the Open Source Revolution »
(O'Reilly Associates, Inc.™) réalisée par des volontaires bénévoles.

Mis en page par www.framasoft.net

Table des matières

Présentation

Édition

Cet ouvrage est l'adaptation française, réalisée par des volontaires bénévoles fédérés dans le cadre d'un projet des éditions O'Reilly mené grâce à l'Internet, de l'édition d'avril 1999 du livre intitulé « Open Sources : Voices of the Open Source Revolution » (ISBN 1-56592-582-3), publiée aux États-Unis par O'Reilly Associates, Inc.™

Contributions

De nombreux, courageux et efficaces volontaires participèrent à la réalisation de la présente adaptation française.

L'éditeur tient à remercier :

quelques-uns des tenaces traducteurs/relecteurs :

> Bernard Choppy, Olivier Cotinat, Sébastien Blondeel, Éric Dumas, Guy Brand, Roger Espel Llima, Laurent Cheylus, Patrick Loiseleur, Laurent Caillat-Vallet, Philippe Malinge, Christelle Reggiani, Stéfane Fermigier, Bernard Niset, Dave Schroeter, Fabrice Noilhan, Sidoine Pierrel, Julien Jolivet, Yann Forget, Albert-Paul Bouillot, Philippe Martin, Jacques Chion, Gilles Lamiral, Joël Sagnes...

quelques-uns des relecteurs pointilleux :

> Bernard Choppy, Jean Zundel, Arnaud Launay, Hubert Figuière, Jérôme Arnaud, Laure Piganiol, Michael Tiemann, Richard Stallman, Thierry Le Queau, Hervé Mignot, Claire Boussard, Pierre Brua, Serge Dieth, Raphael Gurlie, Anne Crouzeix, Nicolas Chauvat, Guillaume Allegre, Sameh Ghane, Didier Belot, Dominique Rousseau, Marc Olivier Huntzinger, Jean-Christophe Fargette, Etienne Cazin, Maxime Vernier...

les auteurs de commentaires et de remarques :

> Maxime Vernier

les princes du SGML :

> Xavier Cazin, Sébastien Blondeel, Nicolas Bonnand et Gérald Doussot.

Remerciements

Chris DiBona

Aucun livre comme celui-ci ne peut voir le jour sans l'aide et le conseil de plusieurs personnes, ainsi je remercie ma maman, mon papa, Trish, Denise, Neil, et Mickey. J'aimerais également remercier toutes les personnes du CoffeeNet, qui ont toujours me réserver le meilleur accueil. Mes remerciements vont aussi, bien entendu, aux différents collaborateurs qui ont gaspillé leur temps précieux, d'ordinaire consacré à la programmation, pour travailler sur ce livre ; je leur en suis reconnaissant !

Merci aux gens de VA Research Linux Systems. Je n'aurais pas pu espérer de personnes plus intelligentes et consciencieuses ; merci de m'avoir aidé durant la gestation de ce livre.

Cet ouvrage n'existerait pas sans le soutien et le dévouement de Mark Stone, qui joua un rôle capital durant son élaboration. Il a déclaré : « un livre pourrait être écrit sur la manière dont celui-ci l'a été », ce qui, à mon sens, est exprimé de la façon la plus favorable. Je suis sûr qu'un jour il y repensera et en rira. N'est-ce-pas Mark ? Mark ?

Durant la phase initiale de réflexion, plusieurs personnes ont apporté des idées et leur soutien. Parmi elles se trouvent Paul Crowley, Paul Russell, Corey Saltiel, Edward Avis, Jeff Licquia, Jeff Knox, Becky Wood, et l'homme dont le site web fut un véritable catalyseur, Rob Malda. Merci à tous, j'espère que ce livre répond à vos attentes.

Enfin, je n'aurais pas pu achever ce travail sans le soutien moral de Christine Hillmer, qui par altruisme a déballé notre appartement seule tandis que je travaillais dur sur mon clavier. Vous êtes tout ce dont dont j'ai pu rêver, et je suis conscient de mon bonheur.

Sam Ockman

Je veux remercier les personnes suivantes qui m'aidèrent de différentes manières : Joe McGuckin, Peter Hendrickson, Jo Schuster, Ruth Ockman, Allison Huynh, et Nina Woodard. Je tiens également à remercier mes hackers préférés : David S. Miller et H. Peter Anvin.

Mark Stone

Je souhaite remercier Sam et Chris pour m'avoir exposé d'emblée cette folle idée. Je suis convaincu qu'ils n'avaient aucune idée de ce qu'elle impliquait et aussi que le résultat justifie nos efforts. Je voudrais aussi remercier chacun des collaborateurs ; chacun est un esprit créatif avec plus d'idées que de temps pour les réaliser, mais tous comprirent l'importance de ce projet et la nécessité de lui consacrer du temps.

La réussite d'un livre doit beaucoup aux efforts de personnes qui font des prodiges en coulisses. Sur cet ouvrage en particulier, certains de ces « héros silencieux » méritent des remerciements : Troy Mott, Katie Gardner, Tara McGoldrick, Jane Ellin, Robert Romano, Rhon Porter, Nancy Wolfe Kotary, Sheryl Avruch, Mike Sierra, et Edie Freedman. J'aimerais bien sûr aussi remercier mes amis et ma famille qui durent endurer les effets d'« un nouveau projet dingue

de papa ». Enfin, mes remerciements vont également à l'aimable personnel du Lytton Coffee Roasting Company, mon « bureau loin de la maison ».

Chapitre 1

Introduction

1.1 Prologue

Le créateur de Linux, Linus Torvalds, raconte que son prénom a été choisi pour lui à cause de l'admiration que ses parents vouaient pour le lauréat du Prix Nobel, Linus Pauling. Pauling était un homme des plus rares : un scientifique qui remporta le Prix Nobel, pas une fois, mais deux fois. Nous retrouvons là un récit édifiant pour la communauté Open Source dans l'histoire du travail fondamental qui rendit possible la découverte de la structure de l'ADN.

La découverte, en réalité faite par Francis Crick et James Watson, est racontée par ce dernier de façon fameuse dans le livre intitulé « The Double Helix ». Ce livre est un compte-rendu remarquable et sincère de la façon dont la science est vraiment pratiquée. Il relate non seulement l'intelligence et la perspicacité, mais aussi la politique, la compétition, et la chance. La quête du secret de l'ADN était devenue une féroce compétition entre diverses entités, en particulier le laboratoire de Watson et Crick à Cambridge, et celui de Pauling à Cal Tech.

Watson décrit avec une gêne évidente la manière dont Pauling en est venu à savoir que Watson et Crick avaient résolu le mystère, et avaient créé un modèle de la structure hélicoïdale de l'ADN. Le personnage central de l'histoire y est en fait Max Delbruk, un ami commun qui voyageait entre Cambridge et Cal Tech. Il comprenait et acceptait le désir de Watson et Crick de garder la découverte secrète tant que tous les résultats ne seraient pas confirmés, Delbruk vouait avant tout allégeance à la science elle-même. Voici comment Watson décrit comment il a su que Pauling avait appris la nouvelle :

Linus Pauling entendit parler pour la première fois de la double hélice par Max Delbruk. À la fin de la lettre qui dévoilait la découverte des chaînes complémentaires, j'avais demandé à ce qu'on n'en parle pas à Linus. J'étais encore quelque peu inquiet que quelque chose ne tourne pas rond et je ne voulais pas que Pauling pense aux paires de base limitées en liaisons hydrogène jusqu'à ce que nous ayons pu nous donner encore quelques jours pour vérifier notre position. Malgré cela, ma demande fut ignorée. Delbruk voulait apprendre la nouvelle à tous ses collaborateurs, et il savait qu'en quelques heures la rumeur se propagerait depuis son laboratoire de biologie vers celui de leurs amis travaillant sous les ordres de Linus. Pauling lui fit donc promettre de lui communiquer l'information qu'il avait entendue de moi. En matière de science Delbruk haïssait le secret quelque forme que ce soit et ne voulait pas laisser plus longtemps Pauling dans l'expectative.

Clairement, le besoin de secret avait rendu Watson mal à l'aise. Il comprenait que la compétition interdisait aux deux parties de partager ce qu'elles savaient, et cela ralentissait le progrès de la science.

La science relève bien de l'Open Source. La méthode scientifique repose sur un processus de découverte et de justification. Pour que les résultats scientifiques soient justifiés, ils doivent être reproductibles. La reproduction n'est possible que si les acteurs partagent leurs sources : l'hypothèse, les conditions de test et les résultats. Le processus de découverte peut suivre de nombreux chemins, et de temps à autres des découvertes scientifiques sont isolées. Mais finalement, le processus de découverte doit être servi par le partage de l'information : permettre à d'autres scientifiques d'avancer là où l'on n'y parvient pas donc diffuser les idées pour laisser émerger les nouvelles.

1.2 Qu'est-ce que le logiciel libre, et en quoi se rapporte-t-il à l'OpenSource ?

En 1984, Richard Stallman, un chercheur au laboratoire d'intelligence artificielle du MIT (MIT AI Lab), démarra le projet GNU. Ce projet visait à faire en sorte que personne ne se trouve jamais tenu de payer pour du logiciel. Stallman le lança parce qu'il sentait essentiellement que le savoir qui constitue un programme exécutable (ce que l'industrie informatique appelle le code source) devrait être libre. S'il ne l'était pas, raisonne-t-il, un petit mais puissant groupe personnes domineraient l'informatique.

Là où les éditeurs de logiciels commerciaux et propriétaires n'ont vu qu'une industrie gardant pour elle des secrets économiques qui doivent être jalousement protégés, Stallman a vu un savoir scientifique qui doit être partagé et distribué. Le principe de base du projet GNU et de la Free Software Foundation (l'organisation gérant le projet GNU) est que le code source accélère le progrès en matière d'informatique car l'innovation dépend de la diffusion du code source.

Stallman s'est préoccupé de la manière dont le monde réagirait au logiciel libre. La connaissance scientifique est souvent dans le domaine public ; c'est une des fonctions de la publication académique de la rendre publique. Avec le logiciel, malgré tout, il était clair que que le simple fait de laisser le code source rejoindre le domaine public aurait tenté le monde des affaires de le détourner à son profit. La réponse de Stallman à cette menace était la « GNU General Public License », connue sous le nom de GPL (voir section 5.10).

La GPL autorise l'utilisateur à copier et distribuer à volonté le logiciel qu'elle protège, pourvu qu'il n'interdise pas à ses pairs de le faire aussi, soit en leur faisant payer pour le logiciel en tant que tel, soit en le plaçant sous une autre licence. La GPL requiert aussi que tout dérivé d'un travail placé sous sa protection soit lui aussi protégé par elle.

Lorsque Stallman et d'autres auteurs de textes publiés dans ce livre parlent du logiciel libre, ils traitent de liberté et non de gratuité. Le terme anglais rend mal cette distinction entre gratuité et liberté, et l'expression *Free as in speech, not as in beer*[1]. Ce message radical a mené beaucoup de sociétés de logiciels à rejeter de façon absolue le logiciel libre. Après tout, leur préoccupation est de gagner de l'argent, non d'ajouter à notre corpus de connaissance. Pour Stallman, ce désaccord entre l'industrie informatique et la science de l'informatique était acceptable, peut-être même désirable.

1.3 Qu'est-ce que le logiciel Open Source ?

Durant le printemps 1997 un groupe de leaders de la communauté du logiciel libre se sont rassemblés en Californie. Ce groupe incluait, entre autres, Eric Raymond, Tim O'Reilly et le président de VA Research, Larry Augustin. Leur souci était de trouver une façon de promouvoir les idées entourant le logiciel libre vers les gens qui avaient autrefois fui le concept. Ils s'inquiétaient des effets du message apparemment hostile à toute perspective mercantile de la Free Sofware Foundation, qui pourrait interdire aux utilisateurs de bénéficier de la puissance du logiciel libre.

1. *free speech* signifie « liberté de parole » et *free beer* « bière gratuite », donc « à l'œil ».

Eric Raymond insista afin que le groupe affirme officiellement qu'une campagne de prosélytisme était nécessaire. De cette discussion, il sortit un nouveau terme pour décrire le logiciel qu'il voulait promouvoir : Open Source. Une série de directives furent conçues pour décrire le type de logiciel qui pourrait être qualifié d'Open Source.

Bruce Perens avait déjà jeté les bases de ce qui allait devenir la définition de l'Open Source. L'un des buts du projet de GNU était de créer un système d'exploitation librement disponible qui pourrait servir de plate-forme pour faire fonctionner les logiciels GNU. Dans un cas classique d'auto-amorçage (bootstrapping) logiciel, LINUX était devenu cette plate-forme, et LINUX avait été créé avec l'aide des outils GNU. Perens avait piloté le projet Debian, qui mit en place une distribution de LINUX incluant seulement du logiciel qui adhérait à l'esprit de GNU. Perens avait imposé la chose explicitement dans un document nommé le « Contrat Social Debian » (Debian Social Contract). L'Open Source Definition est un descendant direct du Debian Social Contract, et donc l'Open Source est en grande partie dans l'esprit de GNU.

L'Open Source Definition permet plus de libertés que la GPL. Elle permet surtout une plus grande promiscuité lors d'un mélange de code propriétaire avec du code open source.

En conséquence, une licence Open Source pourrait éventuellement permettre l'emploi et la redistribution d'un logiciel Open Source sans compensation ni même reconnaissance de mérite. Vous pouvez, par exemple, prendre une grande partie du code source du navigateur Netscape et la distribuer avec un autre programme, éventuellement propriétaire, sans même en aviser Netscape. Pourquoi Netscape souhaiterait-il cela ? Pour un certain nombre de raisons, mais la plus évidente est que cela leur permettrait de prendre une plus grande part de marché pour le code de leur navigateur, qui fonctionne très bien au sein de leur offre commerciale. De cette façon, le fait de donner le code source est une très bonne façon de construire une plate-forme. C'est aussi une des raisons pour lesquelles les gens de Netscape n'ont pas utilisé la GPL.

Ceci n'est pas une mince affaire pour la communauté. À la fin de 1998 une importante dispute menaça de diviser la communauté LINUX. Cette fracture était causée par l'avènement de deux systèmes logiciels, GNOME et KDE, tous les deux avaient pour but de bâtir une interface de bureau orientée objet. D'un côté, KDE utilisait la bibliothèque Qt de Troll Technology, un morceau de code alors propriétaire, mais assez stable et mature. D'un autre côté, les gens de GNOME avaient décidé d'utiliser la bibliothèque GTK+, qui complètement libre mais moins mûre que Qt.

Dans le passé, Troll Technology aurait eu à choisir entre, d'une part, utiliser la GPL et, d'autre part, garder leur position propriétaire. Le désaccord entre GNOME et KDE aurait continué. Avec l'avènement de l'Open Source, néanmoins, Troll était en mesure de modifier sa licence pour qu'elle rencontre la définition de l'Open Source, tout en conservant aux gens de Troll le contrôle qu'ils désiraient sur la technique. Le désaccord entre deux importantes parties de la communauté LINUX semble se résoudre.

1.4 Le côté obscur de la Force

Bien qu'il ne s'en soit probablement pas rendu compte à l'époque, Watson se tenait au seuil d'une ère nouvelle de la science biologique. À l'époque de la découverte de la double hélice, la science biologique et chimique était essentiellement une sorte d'artisanat, un art pratique. Elle était pratiquée par quelques hommes travaillant en petits groupes, principalement sous les auspices de la recherche universitaire. Cependant, les graines du changement avaient déjà été plantées. Avec l'avènement de quelques percées médicales, particulièrement le vaccin contre la poliomyélite et la découverte de la pénicilline, la science biologique était sur le point de devenir une industrie.

Aujourd'hui la chimie organique, la biologie moléculaire, et la recherche médicale fondamentale ne sont plus pratiquées comme un artisanat par un petit corps de praticiens, mais poursuivies par une industrie. Alors que la recherche continue en université, la vaste majorité des chercheurs, ainsi que la vaste majorité des dollars consacrés à la recherche, proviennent de l'industrie pharmaceutique. Cette alliance entre la science et l'industrie est au mieux difficile. Alors que les laboratoires pharmaceutiques peuvent financer la recherche à un niveau inespéré dans une institution universitaire, elles financent aussi les recherches avec un intérêt particulier. Un laboratoire pharmaceutique accorderait-il un budget à la recherche de thérapies plutôt qu'à celle de médications ?

L'informatique, elle aussi, coexiste avec les impératifs d'une l'industrie visant le profit. Il fut une époque où les idées émergeaient principalement d'informaticiens universitaires ; maintenant, c'est l'industrie informatique qui pilote l'innovation. De nombreux programmeurs de l'Open Source sont des étudiants ou des thésards disséminés dans le monde entier, mais un nombre croissant agissent dans un contexte industriel plutôt qu'universitaire.

L'industrie a produit quelques merveilleuses innovations : Ethernet, la souris et l'Interface Utilisateur Graphique (Graphical User Interface, GUI) sont tous issus du laboratoire de Xerox appelé *PARC*. Mais il ne faut pas oublier les aspects inquiétants de l'industrie informatique. Personne en dehors de Redmond [2] ne pense réellement que les gens de Microsoft doivent décider seuls de l'interface graphique ou des possibilités des logiciels.

L'industrie peut avoir un impact négatif sur l'innovation. Le logiciel GNU Image Manipulation Program (GIMP) a langui dans un état incomplet durant une année au niveau d'une version beta 0.9. Ses créateurs, deux étudiants à Berkeley, avaient quitté l'école et embrassé une carrière dans l'industrie, faisant passer leur développement libre en arrière-plan.

1.5 Utilise le Source, Luke

L'Open Source n'est pas une idée imposée par une instance supérieure. Le mouvement Open Source est une révolution authentique car issue du substrat. Des évangélistes comme Eric Raymond et Bruce Perens ont eu un grand succès en changeant le vocabulaire du logiciel libre, mais cette évolution aurait été impossible sans contexte propice. L'une des générations d'étudiants apprirent l'informatique sous l'influence de GNU est maintenant au travail dans l'indus-

2. Ville où se trouve le siège social de Microsoft.

trie, et, depuis des années, y a amené discrètement le logiciel libre par la porte de service. Ils ne le font pas par altruisme mais pour utiliser de meilleurs outils.

Les révolutionnaires sont en place. Ce sont les ingénieurs réseau, les administrateurs de systèmes et les programmeurs qui ont étudié avec les logiciels Open Source durant toutes leurs études, et veulent les utiliser aussi pour évoluer personnellement. Les logiciels libres sont devenus vitaux pour beaucoup d'entreprises, souvent involontairement, mais dans certains cas de façon tout-à-fait délibérée. Le modèle économique de l'Open Source existe et se trouve à présent en phase de maturation.

La société de Bob Young, Red Hat Software, Inc., prospère en faisant cadeau de son produit principal : Red Hat LINUX. Une bonne façon de fournir du logiciel libre est de le conditionner en une distribution complète avec un manuel de bonne facture. Young vend principalement la commodité, parce que la plupart des utilisateurs ne veulent pas télécharger toutes les pièces qui font un système LINUX complet.

Mais il n'est pas seul à proposer cela. Donc pourquoi Red Hat domine-t-il le marché des États-Unis ? Pourquoi SuSE LINUX domine-t-il le marché européen ? Le marché de l'Open Source est similaire à un marché de produits destinés aux grand-public. Dans tous les marchés de ce type le client apprécie une marque à laquelle il accorde sa confiance. La force de Red Hat vient de la gestion de la marque : un marketing cohérent et un contact avec la communauté qui fait que celle-ci les recommande lorsque leurs amis leur demandent quelle distribution utiliser. La même chose est vraie pour SuSE, et les deux entreprises possèdent leurs marchés respectifs principalement parce qu'elles étaient les premières à prendre au sérieux la gestion du nom de la marque.

Soutenir la communauté est essentiel. Red Hat, SuSE, et d'autres sociétés de l'univers LINUX comprennent que faire seulement de l'argent sans donner quoi que ce soit en retour causerait des problèmes.

Premièrement, les gens considéreraient cela comme du parasitage et recommanderaient un concurrent.

Deuxièmement, une société doit être en mesure de se différencier des concurrents.

Diverses entreprises, par exemple CheapBytes et LINUX Central, fournissent une distribution à bas prix, en vendant les CD aussi peu cher qu'un dollar. Pour que Red Hat soit perçue comme offrant une valeur plus importante que ces distributeurs à petit budget elle doit donner quelque chose en retour. Par une superbe ironie du modèle Open Source, Red Hat peut se permettre de vendre sa distribution 49,95 dollars uniquement parce qu'elle en assure le développement et livre le plus gros du code en Open Source.

Ce type de gestion de la marque est nouveau pour l'Open Source. En revanche, l'approche classique qui consiste tout simplement à fournir un bon service, a participé du modèle de l'Open Source. Michael Tiemann a participé à la fondation de Cygnus™ sur une idée simple : bien que le meilleur compilateur du monde, GCC, soit disponible gratuitement, les sociétés souhaiteraient payer un service d'assistance technique et d'amélioration. Le co-fondateur de Cygnus™, John Gilmore, décrit sa société comme apte à : « Rendre le logiciel libre abordable ».

En fait, ce modèle qui consiste à donner un produit et en vendre le support est, maintenant, en train de proliférer rapidement dans le monde de l'Open Source. VA Research a fabriqué et a supporté des systèmes LINUX de grande

qualité depuis fin 1993. Penguin Computing offre des produits et services similaires. LinuxCare propose un service complet, « de l'entrée au dessert » pour Linux dans chacune des variantes. Le créateur de Sendmail, Eric Allman, a maintenant fondé la société Sendmail Inc. ? pour fournir du service et des améliorations autour de son logiciel serveur de messagerie qui détient environ 80 % des parts de marchés. Sendmail est un cas intéressant parce qu'il a une approche double du marché. La société possède la version propriétaire Sendmail Pro, et la version précédente de Sendmail Pro appelée Free Software Sendmail.

Dans la même lignée, Paul Vixie, le président de Vixie Enterprises et coauteur de ce livre, jouit d'un quasi monopole à travers son programme BIND. Ce modeste programme est utilisé chaque fois que vous envoyez un message électronique, que vous visitez un site web, ou que vous téléchargez un fichier à l'aide de ftp. BIND est le programme qui gère la conversion d'une adresse de type « www.dibona.com » vers le numéro IP correspondant (dans ce cas, 209.81.8.245). Vixie vit grâce à son prospère cabinet de conseil, nourri par la diffusion de son programme.

1.6 L'innovation à travers la méthode scientifique

Le développement plus fascinant dans le mouvement Open Source d'aujourd'hui n'est pas le succès de sociétés telles que Red Hat ou Sendmail Inc. Ce qui fascine réellement est de voir les plus importants acteurs de l'industrie informatique, des entreprises telles qu'IBM et Oracle, tourner leur attention vers l'Open Source en y percevant un marché. Que cherchent-elles dans l'Open Source ?

L'innovation.

Là où les scientifiques parlent de reproductibilité, les programmeurs Open Source parlent de déboguage. Là où les premiers parlent de découverte, les seconds voient création. Le mouvement Open Source est une extension de la méthode scientifique, car au cur de l'industrie informatique se trouve la science informatique. Considérez ces mots de Grace Hopper, inventeur du compilateur, qui disait, au début des années 60 :

Pour moi la programmation est plus qu'un art appliqué important. C'est aussi une ambitieuse quête menée dans les tréfonds de la connaissance.

L'informatique, cependant, diffère fondamentalement de toutes les autres sciences. L'informatique a un seul moyen de permettre à des pairs de reproduire des résultats : partager le code source. Pour démontrer la validité d'un programme à quelqu'un, vous devez lui fournir les moyens de le compiler et de l'exécuter.

La reproductibilité rend les résultats scientifiques robustes. Un scientifique ne peut espérer justifier de toutes les conditions de test, ni nécessairement disposer de l'environnement de test qui permettrait de vérifier tous les aspects d'une hypothèse. En partageant les hypothèses et les résultats avec une communauté de pairs, le scientifique permet à de nombreuses paires d'yeux de voir ce qu'une seule pourrait manquer. Dans le modèle de développement Open Source, le même principe est exprimé ainsi : « À condition de disposer de suffisamment de regards, tous les bogues restent superficiels ». En partageant le code source, les développeurs Open Source rendent leur logiciel plus robuste. Les programmes

deviennent utilisés et testés dans une plus grande variété de contextes que ce qu'un seul programmeur pourrait générer, les bugs sont découverts alors qu'autrement ils ne l'auraient pas été. Parce que le code source est fourni, les bugs peuvent souvent être supprimés, et pas seulement découverts, par quelqu'un qui, autrement, serait resté en dehors du processus de développement.

Le partage ouvert des résultats scientifiques facilite les découvertes. La méthode scientifique minimise la duplication des effort parce que les pairs savent d'emblée s'ils travaillent sur des projets similaires. Le progrès ne stoppe pas simplement parce qu'un scientifique cesse de travailler sur un projet. Si les résultats ont de la valeur, d'autres scientifiques prendront le relais. De façon similaire, dans le modèle de développement Open Source, le partage du code facilite la créativité. Les programmeurs travaillant sur des projets complémentaires peuvent appuyer leurs propres résultats sur ceux des autres, ou combiner les différentes ressources dans un seul projet. Un projet peut éveiller l'inspiration pour un autre projet qui n'aurait pas été conçu sans cela. Et un projet de valeur ne devient pas forcément orphelin lorqu'un programmeur le quitte. GIMP est resté au repos durant une année, mais finalement le développement a continué, et maintenant GIMP est montré avec fierté lorsque les développeurs Open Source examinent ce dont ils sont capables dans un domaine complètement nouveau pour eux : les applications utilisateur.

Les plus grosses entreprises veulent tirer profit de ce modèle puissant d'innovation. IBM va faire payer une somme coquette pour mettre en place et administrer l'intégration d'Apache au sein des services informatiques. Il s'agit là d'un gain net pour IBM ; ils peuvent installer une plate-forme exceptionnellement stable, qui réduit le coût assigné à sa maintenance, et, du même coup, assurer un service qui peut réellement aider leurs clients. De façon tout aussi importante, les ingénieurs d'IBM partagent la pollinisation croisée des idées avec les développeurs indépendants de l'équipe d'Apache.

Il s'agit là précisément du raisonnement qui a amené Netscape à prendre la décision de rendre son navigateur Open Source. Une partie du but recherché était de stabiliser et même d'accroître ses parts de marché. Mais, plus encore, Netscape compte sur la communauté des développeurs indépendants pour conduire l'innovation et les aider à créer un produit de qualité supérieure.

IBM s'est rapidement rendu compte que lintégration complète des techniques logicielles telles qu'Apache dans des lignes de plate-forme serveur comme l'AS400 et le RS6000 pouvait au minimum aider à gagner de nouveaux contrats et vendre plus de matériel IBM. IBM va plus loin dans cette approche et porte sa célèbre base de données DB2 vers le système Linux. Alors que beaucoup ont interprété cela comme une réponse à Oracle qui sort la ligne Oracle 8 sous Linux, IBM a pris son rôle envers la communauté très au sérieux et alloua des ressources à la cause des logiciels Open Source. En portant Apache vers la plate-forme AS400, IBM légitime cette approche d'une façon dont cette entreprise reste seule capable.

Le sort réservé aux offres concurrentielles soumises au gouvernement fédéral et à l'industrie par des entreprises telles que Coleman (de Thermo-electron), SAIC, BDM et IBM sera riche d'enseignements. Par exemple lorsque l'on comparera le coût du logiciel nécessaire à l'installation de 1 000 postes avec, d'une part, NT ou Solaris et, d'autre part, Linux. Le prestataire dont la proposition laisse le client économiser plus d'un quart de million de dollars se trouvera en meilleure position. Des entreprises comme CSC, qui ont la réputation de renon-

cer à un ou deux pour-cent de leur marge afin d'obtenir davantage de contrats, sont probablement en train de se demander comment tirer profit de cela.

Alors que des entreprises telles qu'IBM, Oracle et Netscape ont commencé à intégrer l'Open Source à leur approche, beaucoup d'éditeurs de logiciel continuent à se concentrer exclusivement sur des solutions propriétaires. Ils le font à leurs propres risques.

En matière de serveur web, la complète dénégation de Microsoft du phénomène Open Source est presque amusante. Le serveur web Apache a, au moment de la rédaction de ce texte et d'après l'étude Netcraft, plus de 50 % du marché des serveurs. Mais dans la publicité vantant son logiciel serveur appelé Internet Information Server (IIS) Microsoft affirme qu'il possède plus de la moitié du marché des serveurs web il s'agit en réalité de plus de la moitié des serveurs commerciaux. Comparés aux concurrents comme Netscape et Lotus, ils détiennent donc une fraction significative du marché, mais cela semble maigre par rapport au nombre total de serveurs en fonction, où les 20 % de Microsoft sont écrasées par les 50 % d'Apache.

L'ironie est encore plus flagrante encore lorsque l'on considère qu'actuellement, d'après une étude menée par le QUESO et WTF, 29 % des serveurs Web fonctionnent sous Linux. QUESO est un outil logiciel capable de déterminer le système d'exploitation qu'utilise une machine en lui envoyant des paquets TCP/IP et en analysant ses réponses. En combinant ces résultats avec ceux du moteur de recherche Netcraft qui analyse les balises d'identification renvoyées par les serveurs, on peut déterminer le nombre de serveurs employant un système d'exploitation ou un type de machine donné.

En fait, les éditeurs de logiciels propriétaires ont déjà subi de nombreux discrets revers. Linux et FreeBSD rendent impossible de vendre avec succès un Unix propriétaire sur du matériel PC : Coherent, par exemple, s'est déjà effondrée. D'autre part, la société Santa Cruz Operation est passée en deux ou trois ans du rôle de principal vendeur Unix à une position de second plan. Elle va probablement trouver un moyen de survivre, mais son produit phare, SCO Unix, sera-t-il une victime de plus de l'Open Source ?

Sun Microsystems a, de plusieurs façons, fourni des ressources au développement de logiciels Open Source tout au long des années, sous forme de donation de matériel, de ressources pour aider au port sur SPARC de Linux, ou à travers le développement du langage de script Tcl réalisé par John Ousterhout. Il est donc paradoxal que l'entreprise, issue des joyeuses racines du logiciel libre à Berkeley si souvents décrites par Kirk McKusick, se batte à présent pour saisir la portée du phénomène Open Source.

Prenons un moment pour comparer les positions contrastées de SCO et de Sun.

Sun réalise la plus grosse partie de son chiffre d'affaire grâce au service commercialisé portant sur son système d'exploitation et son matériel. Sa ligne de produits va de la station de travail de bureau à des prix comparables à ceux des PC, à des serveurs d'entreprise de grande taille (éventuellement en grappes) rivaux de certaines grosses machines centrales. Le profit que Sun réalise au niveau du matériel provient moins des ventes des machines bas de gamme de la série Ultra que de celles du service d'assistance associé aux serveurs extrêmement spécialisés et adaptables au client des séries E et A. On estime que 50 % du bénéfice de Sun provient de l'assistance, des formations et du conseil.

SCO, d'un autre côté, fait des affaires en vendant le système d'exploitation

SCO Unix, des programmes de type compilateur et serveurs, ainsi que de la formation sur l'utilisation des produits SCO. Ainsi, malgré sa réputation, elle est en danger de la même façon qu'une ferme disposant d'un seul type de culture est vulnérable au moindre parasite menaçant la récolte.

Sun perçoit le développement de LINUX comme une menace pour son bas de gamme. Sa stratégie consiste à s'assurer que LINUX puisse fonctionner sur le matériel Sun, afin de laisser clients choisir. L'intérêt pour Sun est qu'ils pourront continuer à proposer de l'assistance pour leur machines. Nous ne serions pas surpris de voir, dans le futur, Sun offrir, sur ses machines bas de gamme, de l'assistance à l'utilisation du logiciel pour LINUX.

Cela constitue même, et pour plusieurs raisons, une étape évidente. En fait, si vous demandez à un administrateur système sur Sun quelle est la première chose qu'il fait lorsqu'il reçoit une nouvelle machine Sun, il vous répondra : « Je télécharge les outils et le compilateur GNU puis installe mon shell favori ». Sun entendra peut-être un jour ce message provenant de sa clientèle et le fera peut-être sur sa demande même. Malgré cela, Sun connaîtra un certain désavantage jusqu'à ce que ses responsables se rendent compte des services que sa clientèle peut lui apporter : l'innovation à travers la pollinisation croisée des idées issues de la publication du code source.

La position SCO, par ailleurs, est moins avantageuse. Le système de tarification de SCO met en avant la vente du système d'exploitation, en prévoyant des coûts supplémentaires pour des outils que l'utilisateur de LINUX s'attend à recevoir gratuitement, tels que les compilateurs et les systèmes de traitement de texte. Ce modèle ne peut simplement pas être soutenu face à la compétition d'un système d'exploitation robuste et gratuit. À la différence de Sun, qui commercialise une vaste gamme de matériels, SCO ne vend que du logiciel et, dans leur cas, ce n'est pas suffisant. Que va faire SCO ?

Sa réponse n'a, jusqu'à présent, pas été claire. Dans le début de l'année 1998, SCO a envoyé une lettre à la grande liste de diffusion de ses utilisateurs afin de critiquer les Unix ouverts tel que LINUX et FreeBSD en leur reprochant d'être instables et non professionnels, indépendamment de leur prix inférieur à celui du système SCO de base. Ils a lui-mêmes été très critiqué pour cette attitude et a fait machine arrière. SCO n'a pas cru ses clients suffisamment perspicaces pour voir à travers le FUD[3]. SCO a, en fin de compte, publié une rétractation sur son site web.

SCO diffusa fin 1998 un communiqué de presse annonçant que SCO Unix possède maintenant une couche de compatibilité LINUX, de telle façon que vos programmes LINUX favoris peuvent s'exécuter sous SCO Unix. La réponse a été claire et nette. Pourquoi dépenser de l'argent pour un système dans le but de le rendre compatible avec une offre gratuite ?

SCO dispose d'une position avantageuse si elle veut bénéficier du mouvement Open Source car possède une propriété intellectuelle très valable utilisable afin d'acquérir une véritable position de pouvoir dans le futur de l'Open Source. Pour ce faire, cependant, elle doit radicalement changer d'approche. Plutôt que de percevoir l'Open Source en tant que menace qui pourrait éroder la valeur de sa propriété intellectuelle, elle doit y voir une occasion de puiser de l'innovation.

Bien sûr, les manuvres d'une entreprise telle que SCO ou même de Sun sont

3. Fear Uncertainty Doubt : Peur, Incertitude et Doute. Cette expression désigne les campagnes d'intoxication souvent relayées par les media.

insignifiantes à côté de celles de Microsoft qui campe sur sa position en faveur du modèle propriétaire et semble déterminé à ne pas réformer cela avant la publication de MS-Windows 2000.

Nous supposons que MS-Windows 2000 sera annoncé en fanfare dans la dernière partie de l'an 2000 ou au début de 2001. Ce sera, après tout, la grande unification de MS-Windows NT et 98. À un certain moment, aux alentours de cet événement et peut-être même six mois avant, un nouveau système d'exploitation Microsoft Windows sera annoncé. Cet éditeur a toujours convoité le « lucratif marché des entreprises », un secteur où les machines véhiculent la sève informationnelle de l'entreprise. Jusqu'à présent, néanmoins, il n'est pas évident que Microsoft parviendra à fournir un système MS-Windows aussi stable que l'exige ce marché. Ils décréteront donc que ce nouveau système sera la solution.

Appelons ce produit qui représente ce changement fantomatique « MS-Windows Enterprise » ou WEnt. Microsoft se penchera sur NT et déclarera : « Comment pouvons-nous le rendre plus fiable et stable ? ». Comme le signale Linus Torvalds dans son essai, la théorie des systèmes d'exploitation n'a pas évolué au cours des 20 dernières années, donc les ingénieurs de Microsoft vont revenir en arrière et dire qu'un noyau bien écrit, auquel on confie autant que possible l'exécution des programmes en mode non privilégié, est la meilleure façon d'améliorer la fiabilité et les performances. Ainsi pour réparer les principales erreurs des noyaux de NT, par exemple l'inclusion de pilotes réalisés de façon incorrecte par des tierces parties et l'intégration du GUI (Graphic User Interface) au noyau, Microsoft devra soit écrire une monstrueuse et lente couche d'émulation, soit changer d'approche et donc rendre de nombreuses applications incompatibles. Microsoft est certainement capable de choisir l'une des deux voies, et de s'y tenir. Mais pour alors, les logiciels Open Source auront certainement atteint une maturité où les entreprises qui achètent les logiciels se demanderont si elles font encore confiance à Microsoft pour leur donner ce qui est déjà disponible avec LINUX : un noyau stable.

Évidemment, la réponse s'imposera d'elle-même avec le temps. Personne ne sait réellement si Microsoft peut réellement écrire un noyau stable et robuste. Le « document Halloween », auquel Eric Raymond se réfère dans cet ouvrage, laisse entendre que certains salariés de Microsoft en doutent.

1.7 Périls pour l'Open Source

La plupart des éditeurs de logiciels, comme la plupart des entreprises scientifiques, échouent parfois. En matière d'édition de logiciels, rappelle Bob Young, les principes grâce auxquelles on mène l'entreprise vers le succès varient peu de ceux qui profitent à leurs concurrents plus classiques. Dans les deux cas, le réel succès s'avère rare et les meilleurs innovateurs tirent les leçons de leurs erreurs.

La créativité, qui mène à l'innovation tant dans le domaine de la science que dans celui du logiciel, n'est possible qu'à un certain prix. Garder le contrôle sur un projet Open Source dynamique peut s'avérer difficile. Cette peur de perdre le contrôle empêche certaines personnes et beaucoup d'entreprises de fournir une réelle participation. En particulier une inquiétude courante lors de l'adhésion à un projet Open Source est qu'un concurrent ou un groupe de gens y entre également et ne crée ce qui est appelé un embranchement dans le code source. De même qu'un embranchement sur la route, une base de code source peut à

certain moment se diviser en deux routes séparées et incompatibles qui ne se croiseront plus. Ce n'est pas une vaine inquiétude ; observez, par exemple, les multiples embranchements que les systèmes BSD ont empruntés pour mener vers NetBSD, OpenBSD, FreeBSD et de nombreux autres. Seule la méthode ouverte utilisée pour le développement de LINUX garantit qu'il ne subira pas le même sort.

Linus Torvalds, Alan Cox et d'autres développeurs forment une équipe unie et demeurent l'autorité centrale de gestion de l'évolution du noyau. Le mode de gestion du noyau LINUX a été appelé « dictature bon enfant », avec Linus en tant que dictateur éclairé, et il a jusqu'à présent produit un noyau bien écrit et cohérent.

Il est amusant de constater que LINUX a subi extrêmement peu de réels embranchements. Il existe des patches importants qui convertissent le noyau de LINUX en un noyau temp-réel, adapté au contrôle précis de périphériques rapides, et aussi des versions capables de fonctionner au sein d'architectures très inhabituelles. Ces variations peuvent être considérées comme des embranchements dans la mesure où elles concernent une version du noyau et évoluent à partir de là, mais elles occupent des niches d'applications très particulières et n'ont pas un effet de scission sur la communauté LINUX.

Considérons, par analogie, une théorie scientifique appliquée à certains cas particuliers. La plus grande partie du monde se débrouille parfaitement en utilisant les lois de Newton du mouvement pour les calculs mécaniques. Un contexte particulier où entrent en jeu de très grandes masses ou des vitesses élevées nous obligent à recourir à relativité. La théorie d'Einstein pourrait progresser et s'étendre sans ébranler l'application de la base théorique newtonienne plus ancienne.

Mais, bien souvent, des entreprises logicielles compétitives s'opposent, exactement comme certaines théories scientifiques. L'histoire de Lucid, par exemple, en offre un belle illustration. Cette entreprise développait et exploitait une version adaptée de l'éditeur Emacs afin de le vendre à la communauté des développeurs en tant que remplacement de l'Emacs original, écrit par Richard Stallman. Cette version fut appelée Lucid Emacs puis XEmacs. Lorsque l'équipe de Lucid lança XEmacs auprès de différentes entreprises, ils se rendirent compte que les résultats produits grâce à XEmacs n'étaient pas très différents de ceux d'Emacs. La situation terne du marché informatique à cette époque, combinée à ce fait, écourta l'existence de Lucid.

Il est intéressant de constater que Lucid rendit public le code de XEmacs sous GPL. Même l'échec de cette entreprise est une indication de la longévité qu'assure le modèle Open Source aux logiciels concernés. Tant que des gens apprécieront ce logiciel, Lucid fournira la maintenance nécessaire pour qu'il fonctionne sur les nouveaux systèmes et architectures. XEmacs profite d'une extraordinaire popularité, et on peut lancer un débat intéressant parmi les hackers d'Emacs en leur demandant quelle version ils préfèrent. XEmacs, même avec très peu de personnes y travaillant, est encore un produit important et vivant, modifié, maintenu, et peu à peu adapté aux nouvelles époques et architectures.

1.8 La motivation du hacker Open Source

L'expérience de Lucid montre que les programmeurs gardent souvent une loyauté envers un projet qui va plus loin qu'une compensation directe correspondant à la participation au projet. Pourquoi des gens écrivent-ils du logiciel libre ? Pourquoi donnent-ils gratuitement ce qu'ils pourraient facturer des centaines de dollars de l'heure ? Qu'en retirent-ils ?

Leur motivation n'est pas seulement l'altruisme. Ces contributeurs n'ont sans doute pas les poches pleines d'actions Microsoft, mais chacun bénéficie d'une réputation qui devrait lui assurer des occasions leur permettant de payer le loyer et de nourrir leurs enfants. Vu de l'extérieur, cela peut paraître paradoxal car, après tout, on ne peut se nourrir de logiciel libre. La réponse se trouve, en partie, au-delà de la notion de rétribution du travail. Nous somme témoins de la naissance d'un nouveau modèle économique, et non pas seulement d'une nouvelle culture.

Eric Raymond s'est posé en anthropologue participant à la communauté Open Source. Il analysa les motivations de ceux qui développent des logiciels afin d'en faire don par la suite.

Gardez à l'esprit que ces gens ont, pour la plupart, programmé pendant des années, et ne voient pas la programmation en soit comme une chose ennuyeuse ou comme un travail. Un projet très complexe comme Apache ou comme le noyau de Linux apporte une satisfaction suprême sur le plan intellectuel. Un vrai programmeur, après avoir terminé et débogué une morceau de code récursif abominablement difficile qui aura été pour lui une source de problèmes durant plusieurs jours, ressentira une excitation proche de celle que connaît un sportif lorsqu'il participe à une course.

Le fait est que beaucoup de programmeurs écrivent des programmes parce que c'est ce qu'ils aiment faire, et c'est précisément la manière dont ils définissent leur forme d'esprit. Sans le codage (écriture de programme), un programmeur se sent moins complet, comme un athlète privé de compétitions. La discipline peut poser un problème au programmeur tout autant qu'à l'athlète ; beaucoup de programmeurs n'aiment pas maintenir un morceau de code après l'avoir terminé.

D'autres programmeurs n'adoptent toutefois pas ces conceptions de durs à cuire et préfèrent un point de vue plus classique. Ils se considèrent, à juste titre, comme des scientifiques. Un scientifique n'est pas censé amasser des profits grâce à ses invention mais doit publier et partager ses découvertes pour le bénéfice de tous. Il préfère théoriquement la connaissance au profit matériel.

Tout cela procède de la quête d'une réputation. La programmation est une culture du don : la valeur du travail d'un programmeur provient du fait qu'il le partage avec d'autres. Cette valeur est accrue lorsque le travail est mieux partagé, et encore plus lorsqu'il l'est en publiant les sources et non les résultats d'un exécutable précompilé.

La programmation est aussi une prise de pouvoir, ce qu'Eric Raymond désigne par « gratter une démangeaison ». La plupart des projets Open Source prirent leur essor grâce à un sentiment de frustration : après avoir recherché vainement un outil capable d'accomplir une tâche donnée, ou en avoir découvert un qui n'était pas au point, abandonné, ou mal maintenu. C'est de cette façon qu'Eric Raymond commença à développer fetchmail, que Larry Wall amorça le projet Perl, que Linux Torvalds créa Linux. La prise de pouvoir, pour plu-

sieurs raisons, était le concept le plus important sous-jacent à la motivation de Stallman pour démarrer le projet GNU.

1.9 L'avenir du capital-risque et de l'investissement dans Linux

Les motivations du hacker sont parfois philosophiques, mais le résultat n'est pas nécessairement un style de vie tendu vers le sacrifice. Des entreprises et des programmeurs convaincus de l'intérêt de l'Open Source collaborent de façon nouvelle, réaliste et efficace. L'investissement et le capital-risque entraînent l'économie de la Silicon Valley, où nous travaillons et vivons, et il en est ainsi depuis que le transistor commença a être exploité commercialement car les investisseurs saisirent alors l'occasion offerte par les microprocesseurs qui remplaçaient les cartes logiques compliquées.

À tout moment une nouvelle technique attire du capital-risque. L'investisseur ne dédaigne pas les grandes sociétés mais réalise qu'un portefeuille uniquement constitué d'entreprises florissantes ne permettra pas d'atteindre des objectifs financiers ambitieux. Il s'intéresse donc à de nouveaux projets menant à une Offre Publique Initiale[4] après moins de trois ans d'investissements ou à un fructueux rachat par des entreprises telles qu'Oracle ou Cisco.

En 1998, la grande vague de l'Internet retombe, et les ravages des IPO (Initial public Offer) de l'Internet, qui commencèrent avec les débuts fulgurants de Netscape, déclinent. Le rachat de Netscape par America Online marque, de façon très symbolique, la fin de cette ère. Les investisseurs s'intéressent de près aux actions d'entreprises liées à l'Internet, et bien souvent les soumettent aux mêmes contraintes que les autres : il leur faut exhiber de plausibles prévisions de profits.

Où le capital-risque mène-t-il ? Linux et les sociétés logiciels liées à l'Open Source, selon nous, sont et resteront le principal thème d'investissement de cette fin de millénaire. Linux, en ce cas, déchaînera l'enthousiasme et une IPO portant sur Red Hat Software à la fin de 1999 déclenchera une vague d'actions financières de ce type. Le volume des fonds disponibles est énorme, et des entreprises telles que Scriptics, Sendmail, et Vix.com semblent bien placées pour tirer parti de conditions favorables. Il ne s'agit plus de déterminer sir le capital-risque investira dans l'Open Source, mais plutôt de déterminer pourquoi le flot a commencé à s'écouler dans cette direction. N'oubliez pas que le logiciel libre n'est pas nouveau ; Richard Stallman a créé la Free Software Foundation en 1984, et elle repose sur une tradition plus ancienne encore. Pourquoi a-t-il fallu si longtemps pour en arriver là ?

En regardant le paysage informatique, on trouve un contexte où une très grande entreprise aux poches très profondes se taille la part du lion du marché commercial. Dans la Silicon Valley, les éditeurs de logiciels enthousiastes cherchent fortune et s'adossent donc à des sociétés de capital-risque. Ils apprennent rapidement que les investisseurs ne s'intéresseront pas à leur projets si ces derniers ne correspondent pas à ceux de Microsoft. Chaque startup doit choisir de jouer le jeu de Microsoft ou de ne pas jouer du tout.

4. IPO : Initial Public Offer.

L'insolente montée en puissance du logiciel libre commença dans cet environnement oppressif. N'importe quel programmeur qui a déjà réalisé un programme destiné au système d'exploitation MS-Windows décrira ce système comme une lourde et démotivante collection d'interfaces construites de façon a rendre le programme complètement dépendant des bibliothèques de Microsoft. Le nombre d'interfaces proposées au programmeur est destiné à rendre n'importe quel programme pour MS-Windows difficile à porter sur un autre système d'exploitation.

L'Internet, domaine que Microsoft ne domine pas encore, ne connaît pas cette restriction. Comme l'indique Scott Bradley, l'Internet est bâti sur un ensemble de conventions et normes ouvertes maintenues par des individus décidés et non par le portefeuille d'une entreprise. Il reste, par bien des aspects, l'aventure originelle de l'Open Source. Le fait de continuer de l'adosser à des standards ouverts a facilité le travail d'une grande diversité de programmeurs sur les applications Internet dont la spectaculaire croissance montre l'efficacité du modèle.

Les structures inhérentes au succès de l'Internet sont également présentes dans le mouvement de l'Open Source. Oui, des distributeurs LINUX comme Red Hat et SuSE sont concurrents, mais avec des standards ouverts et du code partagé. Toutes deux utilisent le Red Hat Package Manager (RPM) comme outil de gestion des paquetages, par exemple, au lieu d'essayer de capturer les développeurs par des systèmes de paquetages propriétaires. Debian emploie un outil différent, mais comme les outils de Debian et de Red Hat sont des programmes ouverts, la compatibilité a été assurée. L'infrastructure qui a rendu l'Internet attrayant pour le capital-risque est donc présente dans l'Open Source et rendra ce dernier tout aussi attirant.

L'Internet, et c'est plus important encore, a créé une nouvelle infrastructure à partir de laquelle l'Open Source peut se déployer. Nous passons de l'ère des éditeurs commerciaux de logiciels à celle des entreprises d'information que décrit Tim O'Reilly. Ce passage exige l'abaissement drastique des coûts d'entrée et de distribution. C'est ce qu'a permis l'Internet.

1.10 Science et Nouvelle Renaissance

Le modèle de développement Open Source actuel prend ses racines dans les sciences informatiques vieilles d'au moins une décennie. Ce qui fait le succès grandissant de l'Open Source aujourd'hui, cependant, est la dissémination rapide de l'information rendue possible par l'Internet. Quand Watson et Crick ont découvert la double hélice, ils pouvaient espérer que l'information circule de Cambridge à Cal Tech en quelques jours, quelques semaines au pire. Aujourd'hui, la transmission d'une telle information est effectivement instantanée. L'Open Source est née dans la « Renaissance numérique » rendue possible par l'Internet, tout comme les sciences modernes sont apparues pendant la Renaissance grâce à l'invention de l'imprimerie.

Durant le Moyen-Âge aucune infrastructure informationnelle efficace n'existait. Les écrits devaient être copiés à grands frais, et, pour cette raison même, devaient transmettre une valeur instantanée. Seule l'information concise et à l'importance patente était transmise, par exemple les transactions commerciales et financières, la correspondance diplomatique. La priorité des écrits spéculatifs des alchimistes, prêtres et philosophes (qui seraient plus tard appelés scientifiques) restait moindre, et ces informations circulaient donc moins vite. L'im-

primerie a changé tout cela en réduisant le coût d'entrée dans l'infrastructure informationnelle. Les savants, qui auparavant travaillaient dans l'isolement, pouvaient pour la première fois établir une communauté avec leurs pairs dans toute l'Europe. Mais cet apprentissage a requis un partage ouvert de l'information.

La notion de liberté académique et le processus que nous appelons maintenant la Méthode Scientifique émergèrent. Rien de ceci n'aurait pu être possible sans la nécessité de former une communauté, et l'ouverture de l'information a, durant des siècles, été le ciment qui maintenu l'unité de la communauté scientifique.

Imaginez un instant que Newton ait gardé secret ses lois de la mécanique et soit entré en affaires comme sous-traitant de l'artillerie pendant les 30 années qui suivirent. « Non, je ne révélerai rien des trajectoires paraboliques, mais peux régler vos canons si vous réglez mes notes d'honoraires ». L'idée elle-même semble absurde. Non seulement la science n'a pas évolué dans ce sens, mais elle n'aurait pas pu évoluer ainsi. Si cela avait était la façon de penser des scientifiques, leurs secrets eux-mêmes auraient empêché la science de se développer et d'évoluer.

L'Internet est l'imprimerie de l'ère numérique. Une fois encore, l'accès à l'infrastructure a été rendu plus facile. Le code source n'a plus besoin d'être distribué sur des rouleaux de papiers comme l'était la version originale d'Unix, ou sur disquettes, comme aux premiers jours de MS-DOS, ou même sur CD-ROM. N'importe quel serveur FTP ou HTTP (Web) peut servir de point de distribution peu coûteux et toujours disponible.

Bien que cette Renaissance véhicule de grandes promesses, nous ne devons pas oublier l'héritage scientifique de plusieurs siècles auquel est adossé le modèle de développement de l'Open Source. La science de l'information et l'industrie informatique coexistent dans une inconfortable alliance aujourd'hui. Les géants de l'industrie comme Microsoft exercent une forte pression visant à préserver le secret des nouveaux développements afin d'augmenter le gain financier à court terme. Mais de plus en plus de réalisations naissent dans l'industrie plutôt qu'à l'université, et l'industrie doit donc veiller à nourrir la discipline au travers du libre partage des idées, c'est-à-dire par le modèle de développement Open Source. Elle assurerait cela sans la moindre motivation altruiste de servir une grande cause, mais pour une raison pragmatique des plus simples : l'intéressement personnel exacerbé.

Tout d'abord, les industriels informatiques font preuve d'étroitesse d'esprit s'ils croient que le revenu financier est l'objectif principal des meilleurs programmeurs de l'Open Source. Pour impliquer ces derniers dans l'industrie, leur propriété doit être respectée. Ils sont impliqués dans une quête de réputation, et l'histoire montre que la renommée issue de créations intellectuelles survit mieux que celle des réussites financières. Nous connaissons tous les noms de quelques grands industriels de ces cent dernières années : Carnegie, Rockefeller, mais la réputation des scientifiques ou inventeurs marquants de la même période dépasse les leurs : Einstein, Edison... Pauling. Au siècle prochain certains sauront peut-être qui était Bill Gates, mais qui se souviendra des autres industriels de l'informatique ? Il est plus probable que les noms de Richard Stallman et Linus Torvalds demeureront connus.

Ensuite, et surtout, l'industrie a besoin de l'innovation scientifique. L'Open Source peut développer et corriger des logiciels à la vitesse de la création scientifique. L'industrie informatique a besoin de la prochaine génération d'idées issues

du développement Open Source.

L'exemple de LINUX, une fois encore, est révélateur. Ce projet a été conçu à peu près cinq années après que Microsoft ait commencé le développement de MS-Windows NT. Microsoft a dépensé des dizaine de milliers de jours-hommes et des millions de dollars dans le développement de son produit. Aujourd'hui LINUX est considéré comme un concurrent compétitif de MS-Windows NT en tant que serveur sur plate-forme PC, et les principaux logiciels correspondants sont portés par Oracle, IBM, et d'autres grands éditeurs. La quantité de logiciels développés selon le modèle Open Source. Seule une entité aux ressources colossales, par exemple Microsoft, pourrait en réaliser autant.

Pour étayer la Renaissance numérique nous avons besoin de l'approche Open Source. Elle amène le progrès non seulement dans le domaine de l'informatique théorique, mais aussi dans le secteur économique de l'informatique.

Chapitre 2

Une brève histoire des hackers

2.1 Prologue : les Vrais Programmeurs

Au commencement étaient les Vrais Programmeurs[1].

Ce n'est pas le nom qu'ils se donnaient. Ils ne se considéraient pas non plus comme des « hackers », et n'avaient pas le sentiment d'avoir créé une caste. Le sobriquet « Vrai Programmeur » est d'ailleurs apparu après 1980. Mais à partir de 1945, les techniques de l'informatique ont attiré la majorité des esprits plus brillants et les plus créatifs du monde. Sur la trace de l'ENIAC de Eckert et Mauchly, on a vu se développer, de manière plus ou moins continue, une culture technique émergente de programmeurs enthousiastes, qui écrivaient du logiciel pour le plaisir.

Le Vrai Programmeur type était un ingénieur ou un physicien. Il portait des chaussettes blanches, des chemises et des cravates en polyester, chaussait des lunettes épaisses et codait en langage machine, en langage d'assemblage, en FORTRAN et en une demi-douzaine de langages aujourd'hui oubliés. Ils étaient les précurseurs de la culture des hackers, les héros trop méconnus de sa préhistoire.

De la fin de la deuxième guerre mondiale au début des années 70, dans les grands jours de la programmation par lots et des « gros systèmes »[2], les Vrais Programmeurs représentaient la culture technique dominante dans le milieu des informaticiens. Certaines portions du folklore vénéré des hackers remontent à cette époque, comme la célèbre histoire de Mel (dont traite le document intitulé Jargon), quelques listes de lois de Murphy, et l'affiche « Blinkenlights », qui se moque des Allemands, et qu'on trouve encore dans de nombreuses salles d'ordinateurs.

Certains des invidus qui baignèrent dans la culture des « Vrais Programmeurs » sont restés actifs jusque dans les années 1990. SeymourCray, concepteur de la lignée Cray de super-ordinateurs, a la réputation d'avoir programmé un système d'exploitation complet de son cru, pour un ordinateur qu'il avait créé. En octal. Sur de simples interrupteurs. Sans faire une seule erreur. Et cela fonctionna. Le Vrai Programmeur suprême.

Plus discret, StanKelly-Bootle, auteur du The Devil's DP Dictionary (dictionnaire de l'informatique du diable, McGraw-Hill, 1981) et chroniqueur hors-pair du folklore des hackers, a programmé en 1948 sur le Manchester Mark I, premier ordinateur utilisable capable de stocker les programmes de façon numérique. De nos jours, il tient des rubriques techniques humoristiques dans des magazines traitant d'informatique, souvent sous la forme d'un dialogue, vigoureux et entendu, avec les hackers d'aujourd'hui.

D'autres, comme David E. Lundstrom, ont couché sur le papier les anecdotes de ces vertes années (A Few Good Men From UNIVAC, 1987[3]).

On doit à la culture des « Vrais Programmeurs » la montée de l'informatique interactive, des textes traitant d'informatique maintenant classiques, et des réseaux. Ils ont donné naissance à une tradition d'ingénierie continue qui devait déboucher, à terme, sur la culture du hacker de logiciel libre d'aujourd'hui.

1. Certains traducteurs rendent cette note humoristique par l'expression : « les Véritables ».
2. *big iron* signifie « gros ordinateurs ».
3. « Des hommes d'honneur chez UNIVAC ».

2.2 Les premiers hackers

On peut placer le point de départ de la culture des hackers, telle qu'on la connaît, en 1961, l'année où le MIT[4] a fait l'acquisition du premier PDP-1. Le comité Signaux et puissance du club de modèles réduits ferroviaires de cet établissement[5] éleva la machine au rang de jouet technique favori et inventa des outils de programmation, un jargon, et toute une culture associée, dont on trouve encore de nombreuses traces aujourd'hui. Ces premières années sont contées dans la première partie du livre Hackers, écrit par SteveLevy (Anchor/ Doubleday, 1984).

Il semble que l'on doit à la culture informatique du MIT la première adoption du terme « hacker ». Les hackers du TMRC ont formé le noyau du laboratoire d'intelligence artificielle (IA) du MIT, locomotive mondiale en matière de recherche en IA au début des années 1980. Et leur influence s'est répandue bien plus loin après 1969, la première année d'activité de l'ARPAnet.

L'ARPAnet était le premier réseau informatique transcontinental à haut débit. Construit par le Ministère de la Défense afin d'expérimenter les communications numériques, il grossit et interconnecta des centaines d'universités, de fournisseurs de l'armée, et de laboratoires de recherche. Il a permis à tous les chercheurs d'échanger des informations avec une vitesse et une souplesse inégalées jusqu'alors, donnant un coup de fouet au travail collaboratif et augmentant énormément l'intensité et la fréquence des avancées techniques.

Mais l'ARPAnet a eu également un autre effet. Ses autoroutes électroniques ont réuni des hackers de tous les États-Unis d'Amérique en une masse critique. Ces derniers, au lieu de demeurer dans des groupes isolés qui développaient autant de cultures propres et éphémères, se sont découvert (ou réinventé) une tribu de réseau.

Les premières manifestations intentionnelles de la culture des hackersles premières listes de jargon, les premières satires, les premières discussions timides de l'éthiquefurent toutes propagées sur l'ARPAnet dans ses jeunes années (la première version du fichier Jargon, pour citer un exemple majeur, date de 1973). La culture des hackers s'est développée dans les universités connectées au réseau, et en particulier (mais pas exclusivement) dans leurs sections d'informatique.

Le MIT fut le laboratoire d'IA où cette culture naquit à la fin des années 1960, et celui de l'université de Stanford (SAIL) puis celui de Carnegie-Mellon (CMU) jouèrent peu à peu un rôle comparable. Ces trois centres florissants pour l'informatique et la recherche en IA attiraient des gens brillants, qui ont énormément contribué à la culture des hackers, tant d'un point de vue technique que folklorique.

Pour comprendre les développements suivants, il nous faut examiner de plus près les ordinateurs eux-mêmes, car la montée et la chute du Laboratoire furent toutes deux dues à des vagues de changements de la technique informatique.

Depuis l'époque du PDP-1, la destinée de la culture des hackers restait liée à la série de mini-ordinateurs PDP de la société Digital Equipment Corporation. Cette société a ouvert la voie de l'informatique interactive commerciale et des systèmes d'exploitation à temps partagé. Leurs machines étant souples,

4. Institut de Technologie du Massachusetts, l'une des universités les plus prestigieuses des États-Unis d'Amérique.

5. « MIT Tech Model Railroad Club », ou « TMRC ».

puissantes, et relativement bon marché. De nombreuses universités s'en procurèrent.

La culture des hackers se développa grâce aux systèmes peu coûteux utilisables par plusieurs personnes simultanément et l'ARPAnet, durant la majeure partie de son existence, fut principalement constitué de machines DEC dont la plus puissante alors, le PDP-10, sortit en 1967. Le 10 resta la machine préférée des hackers pendant près de quinze ans ; et on se rappelle encore avec tendresse et nostalgie de son système d'exploitation TOPS-10 et de son langage d'assemblage MACRO-10, qui occupent une place de choix dans le jargon et dans le folklore des hackers.

Les chercheurs du MIT, qui utilisaient le PDP-10 comme tous leurs pairs dans les autres centres, ont choisi une voie légèrement différente. Ils ont complètement rejeté le logiciel que la société DEC proposait pour cette machine et employèrent leur propre système d'exploitation, le légendaire ITS.

ITS signifie « Incompatible Timesharing System » (système à temps partagé incompatible), ce qui donne une bonne idée de leurs dispositions d'esprit. Ils voulaient travailler à leur manière mais étaient aussi intelligents qu'arrogants. ITS, capricieux, excentrique, et parfois (si pas toujours) bogué, renfermait toute une série d'innovations techniques brillantes, et on peut soutenir, aujourd'hui encore , que c'est le système à temps partagé qui détient le record de la plus longue durée d'exploitation en continu.

ITS lui-même avait été écrit en langage d'assemblage, mais de nombreux sous-projets ont été écrits en langage LISP. Ce dernier était de loin plus puissant et plus souple que tout autre langage de son temps ; en fait, il tient toujours la dragée haute à la plupart des langages d'aujourd'hui car reste, vingt-cinq ans plus tard, mieux conçu. Grâce à lui les hackers de l'ITS réfléchirent de façon nouvelle et créative. C'était l'un des facteurs principaux de leur réussite, et il demeure l'un des langages favoris des hackers.

On utilise encore aujourd'hui de nombreuses créations techniques de la culture d'ITS ; l'éditeur Emacs est probablement l'exemple le plus connu. Le folklore rattaché à ITS reste encore très « vivant » au sein de la communauté des hackers, comme on peut le constater dans le Jargon File.

SAIL et CMU étaient eux aussi très actifs. De nombreux hackers importants qui mûrirent autour du PDP-10 de SAIL devinrent d'éminentes personnalités du monde de l'ordinateur personnel et des interfaces utilisateur à base de fenêtres, d'icônes et de souris employées aujourd'hui. Les hackers de CMU, eux, travaillaient sur ce qui mènerait aux premières applications pratiques à grande échelle de systèmes experts et de la robotique industrielle.

Le Xerox PARC, célèbre centre de recherche installé à Palo Alto, a lui aussi joué un rôle important dans la culture des hackers. Pendant plus de dix ans, du début des années 1970 au milieu des années 1980, il produisit un nombre ahurissant d'innovations révolutionnaires, tant au niveau du matériel qu'au niveau du logiciel. C'est là que les interfaces modernes, à base de souris, de fenêtres, et d'icônes, ont été mises au point. On y a inventé l'imprimante laser, et le réseau local (LAN) ; et les machines de la série D du PARC laissaient présager, avec dix ans d'avance, les puissants ordinateurs personnels du milieu des années 80. Malheureusement, ces génies n'étaient pas prophètes en leur propre société ; à tel point qu'on a pris l'habitude de plaisanter en décrivant le PARC comme un lieu caractérisé par le fait qu'on y développait de brillantes idées. . .pour les autres. Ils influencèrent cependant les hackers de manière décisive.

Les cultures de l'ARPAnet et du PDP-10 se sont renforcées et diversifiées tout au long des années 1970. Les listes de diffusion par courrier électronique, jusqu'alors réservées à des groupes étalés sur des continents entiers intéressés par un thème donné, commencèrent à être utilisées dans des buts plus sociaux et récréatifs. La DARPA [6] ferma délibérément les yeux sur toutes ces activités annexes pourtant « non autorisées » ; car elle avait compris que la très faible surcharge induite était un faible prix à payer pour attirer toute une génération de brillants jeunes gens vers l'informatique.

La plus connue des listes de diffusion à caractère « social » d'ARPAnet était peut-être la liste SF-LOVERS, qui abritait les férus de science-fiction ; elle est toujours bien vivante aujourd'hui, sur l'« Internet », réseau un peu plus grand, héritier de l'ARPAnet. Mais de nombreuses autres listes existaient, ouvrant la voie à un style de communication plus tard commercialisé par des services de temps partagé à but lucratif, tels que les sociétés CompuServe, GEnie, et Prodigy.

2.3 La montée en puissance d'Unix

Pendant ce temps, au plus profond de l'État du Nouveau Jersey, un projet qui allait faire de l'ombre à la tradition du PDP-10, s'animait peu à peu depuis 1969. C'est l'année même de la naissance de l'arpanet qu'un hacker des laboratoires Bell nommé KenThompson inventa Unix.

Thompson avait participé au développement d'un système d'exploitation à temps partagé appelé Multics, qui partageait avec ITS des ancêtres communs. Multics fut un banc de tests pour des idées importantes, comme la manière dont on pouvait dissimuler la complexité d'un système d'exploitation au cur de ce dernier, sans rien en laisser transparaître à l'utilisateur ni même à la plupart des programmeurs. Cela facilitait l'utilisation et la programmation et augmentait donc la proportion de travail consacré à la résolution des problèmes posés et non de ceux qu'induisent l'ordinateur.

Les laboratoires Bell se sont retirés du projet quand Multics a montré des signes de surcharge pondérale (ce système a plus tard été mis sur le marché par la société Honeywell mais n'a jamais connu le succès). Ken Thompson regrettait l'environnement de Multics, et a commencé, sans objectif sérieux, à implanter sur un DEC PDP-7 qu'il avait sauvé du rebut un mélange des concepts gouvernant Multics et de certaines de ses propres idées.

DennisRitchie, un autre hacker, avait inventé un nouveau langage, le « C », pour que Thompson puisse l'utiliser dans son embryon d'Unix. Tous deux étaient conçus pour être agréables, sans contraintes, et souples. Aux laboratoires Bell, le mot a circulé, et ces outils attirèrent l'attention jusqu'à être renforcés, en 1971, par une prime accordée à Thompson et Ritchie afin qu'il réalisent ce que l'on appellerait maintenant un système spécifique de gestion d'activités liées à la production de documents. Mais Thompson et Ritchie visaient de plus grands honneurs.

Traditionnellement, les systèmes d'exploitation avaient été écrits en langage d'assemblage, ardu, pour fonctionner le plus rapidement possible sur leurs machines hôtes. Thompson et Ritchie furent parmi les premiers à comprendre que

6. http://www.darpa.mil/, autorité militaire commanditaire du réseau.

le matériel et les techniques de compilation avaient fait suffisamment de progrès pour permettre d'écrire tout un système d'exploitation en langage C, et en 1974 tout l'environnement avait été porté avec succès sur plusieurs machines de types différents.

Cela n'avait jamais été réalisé auparavant, et les implications étaient énormes. Si Unix pouvait présenter le même visage et les mêmes possibilités sur des machines différentes il pourrait leur servir d'environnement logiciel commun. Cela affranchirait les utilisateurs des coûts de modification des logiciels liée à l'obsolescence des machines. Les hackers pourraient transporter des boîtes à outils logicielles d'une machine à l'autre, plutôt que de devoir réinventer la roue et l'eau chaude à chaque fois.

Unix et C avaient d'autres atouts dans leur manche, et pas des moindres. Tous deux avaient été construits en suivant la philosophie du « Keep It Simple, Stupid » (acronyme donnant KISS et dont la version développée conseille de faire les choses simplement, sans prétentions). Un programmeur pouvait facilement apprendre la totalité de la structure logique du C (à la différence de la plupart des autres langages, antérieurs ou postérieurs) sans devoir se référer sans cesse à des manuels ; et Unix était une sorte de boîte à outils de programmes simples mis au point dans le but de se combiner utilement les uns avec les autres.

Ces combinaisons se révélèrent adéquates pour une large gamme de tâches informatiques, à la plupart desquelles leurs concepteurs n'avaient même pas songé. Le nombre de machines sous Unix exploitées par la société ATT augmenta rapidement malgré l'absence de soutien officiel. En 1980 il avait gagné de nombreux sites informatiques d'universités et de pôles de recherche, et des milliers de hackers en faisaient leur environnement de travail privilégié.

Le PDP-11 et les VAX, ses descendants, étaient les chevaux de labour de la culture Unix des premières années. Mais Unix étant portable, il pouvait fonctionner quasiment à l'identique sur un grand nombre de machines connectées à l'ARPAnet. Et personne n'utilisait de langage d'assemblage car les programmes développés en C étaient facilement portés d'une machine à l'autre.

Unix disposait même, en quelque sorte, de son propre protocole réseau nommé UUCP, lent et (alors) peu fiable mais peu coûteux. Deux machines Unix quelconques pouvaient s'échanger du courrier électronique point à point grâce à des lignes de téléphone ordinaires ; cette fonctionnalité était construite dans le système, ce n'était pas une extension facultative. Les sites Unix ont commencé à former un réseau dans le réseau, et une culture spécifique. 1980 vit la première mouture de l'Usenet, réseau qui dépasserait bientôt l'ARPAnet.

Certains sites Unix se trouvaient eux-mêmes sur l'ARPAnet. Les cultures PDP-10 et Unix se rencontrèrent et à se mêlèrent, mais ce n'était pas toujours heureux. Les hackers PDP-10 avaient tendance à considérer les gens d'Unix comme une bande de parvenus, qui utilisaient des outils d'allure ridicule et primitive si on les comparait aux adorables complexités baroques de LISP et d'ITS. « Couteaux de silex et peaux de bêtes ! », murmuraient-ils.

Il existait encore un troisième courant. Le premier ordinateur personnel avait été mis sur le marché en 1975. La société Apple fut fondée en 1977, et les avancées ont suivi à un rythme effréné et incroyable dans les années qui ont suivi. Le potentiel des micro-ordinateurs était patent et attira une autre génération de jeunes hackers brillants. Ils utilisaient le langage BASIC, qui était si primitif que les partisans de PDP-10 comme les aficionados d'Unix le jugeaient indigne de leur mépris même.

2.4 La fin du bon vieux temps

Telle était la situation en 1980 : trois cultures se recouvraient en partie mais étaient organisées autour de techniques bien distinctes. La culture ARPAnet / PDP-10, vouée au LISP, au MACRO, au TOPS-10, et à ITS. Les gens d'Unix et du C, forts de leurs PDP-11, de leurs VAX, et de leurs connexions téléphoniques rudimentaires. Et une horde anarchique d'enthousiastes des premiers micro-ordinateurs, déterminés à voler aux autres le pouvoir de faire de l'informatique.

Parmi ces cultures, celle de l'ITS pouvait s'enorgueillir de sa position dominante. Mais l'orage menaçait, et les nuages s'accumulaient au-dessus du Laboratoire. Les techniques utilisées dans le PDP-10 vieillissaient, et des factions divisèrent les membres du Laboratoire au cours des premières tentatives de commercialisation des techniques de l'IA. Certains, parmi les meilleurs du Laboratoire (et du SAIL et de CMU) ont succombé aux sirènes d'un emploi très lucratif au sein d'une nouvelle société commerciale.

Le coup de grâce est venu en 1983, quand la société DEC a renoncé à la gamme PDP-10 pour se concentrer sur les modèles PDP-11 et VAX. ITS ne pouvait y survivre. Puisqu'il n'était pas portable, il aurait fallu déployer plus d'efforts que quiconque ne pouvait se le permettre pour le porter sur les nouveaux matériels. La variante d'Unix de Berkeley, qui fonctionnait sur un VAX, est dès lors devenue le système d'élection des hackers, et quiconque gardait un il fixé sur l'avenir devinait que l'augmentation rapide de la puissance des micro-ordinateurs leur permettrait vite de tout balayer.

C'est autour de cette époque que Levy a rédigé le livre « hackers ». L'une de ses sources privilégiées fut Richard M. Stallman (l'inventeur d'Emacs), chef de file au Laboratoire, et le plus féroce opposant à la commercialisation des techniques mises au point par le Laboratoire.

Stallman (repéré généralement par ses initiales RMS, qui forment aussi son nom de compte utilisateur) n'en resta pas là : il créa la Fondation du logiciel libre (FSF) et se consacra à la production de logiciel libre de première qualité. Levy en fait le panégyrique en le présentant comme « le dernier véritable hacker », affirmation qui s'avéra fort heureusement inexacte.

Le grand projet de Stallman illustrait joliment la transition vécue par la culture des hackers au début des années 80. En 1982, il a entrepris le développement d'un clone complet d'Unix, écrit en C et librement disponible. Ainsi, on retrouvait l'esprit et la tradition de l'ITS dans une grande partie de la nouvelle culture des hackers, centrée autour d'Unix et des VAX.

C'est aussi à cette époque que les microprocesseurs et les réseaux locaux ont commencé à avoir un impact considérable sur la culture des hackers. L'Ethernet et le microprocesseur Motorola 68000 formaient un très puissant tandem, et plusieurs sociétés se constituèrent afin de construire la première génération de ce qu'on appelle de nos jours des stations de travail.

En 1982, un groupe de hackers Unix de Berkeley fonda la société Sun Microsystems car ils croyaient qu'un système Unix fonctionnant sur du matériel relativement bon marché à base de 68000 formerait une combinaison gagnante dans une vaste gamme d'applications. Ils avaient raison, et leur projet posa la première pierre de toute une industrie. Les stations de travail, alors trop coûteuses pour la plupart des particuliers, étaient bon marché pour les sociétés et pour les universités. Les réseaux de stations de travail (une par utilisateur) rem-

placèrent rapidement les VAX et autres systèmes à temps partagé, plus anciens.

2.5 L'ère de l'Unix propriétaire

À partir de 1984, au moment du démembrement de la société ATT et alors qu'Unix devenait pour la première fois un produit commercial, l'essentiel de la culture des hackers résidait au sein d'une « nation réseau » relativement cohérente, centrée autour de l'Internet et de l'Usenet (dont la plupart des membres utilisaient un mini-ordinateurou des stations de travail sous Unix), avec un vaste arrière-pays d'enthousiastes de la micro-informatique.

Les stations de travail proposées par Sun et divers autres constructeurs, conçues pour proposer des graphiques de grande qualité et partager les données grâce au réseau, ouvraient de nouveaux horizons aux hackers. Dans les années 1980, les hackers s'intéressaient aux défis posés par la recherche du meilleur mode d'exploitation de ces ressources. L'Unix de Berkeley fournissait les protocoles d'ARPAnet, donc une solution au problème du réseau qui limitait la croissance de l'Internet.

Plusieurs logiciels tentèrent de résoudre le problème posé par l'avénement du graphisme sur des stations de travail. Le système X Window s'est imposé. Le fait que ses développeurs, suivant en cela l'éthique des hackers, souhaitaient mettre gratuitement à disposition de tous le code source de leur solution fut un critère déterminant dans sa réussite ; et c'est l'Internet qui a facilité cette distribution. La victoire de X sur les systèmes graphiques propriétaires (notamment celui que proposait la société Sun) était un présage important de changements qui, quelques années plus tard, affecteraient profondément le système Unix lui-même.

La rivalité ITS/Unix survivait encore par quelques dissensions qui surgissaient à l'occasion (souvent du fait d'anciens partisans du système ITS). Mais la dernière machine employant ITS fut arrêtée pour de bon en 1990 ; les zélateurs n'avaient plus rien à défendre et se sont pour la plupart intégrés à la culture Unix, en grommelant plus ou moins.

Pour les hackers connectés au réseau la grande rivalité des années 1980 opposait les défenseurs du système Unix de Berkeley aux versions proposées par la société ATT. On trouve encore des exemplaires d'une affiche de l'époque qui représente à la manière d'une bande dessinée un vaisseau spatial de combat aux ailes en X (comme ceux qu'on trouve dans la trilogie « La guerre des étoiles », très populaire parmi les hackers) filant à toute allure pour s'éloigner d'une Étoile de la Mort en train d'exploser, et couverte du logo de la société ATT. Les hackers de Berkeley aimaient se considérer comme des rebelles s'opposant aux empires d'entreprises commerciales dépourvues d'âmes. L'Unix ATT n'a jamais rattrapé BSD/Sun en termes de parts de marché, mais il a gagné la guerre des standards. En 1990, les versions d'ATT et de BSD étaient devenues plus difficiles à distinguer, chacune ayant beaucoup emprunté à l'autre.

Au début des années 1990, les capacités des stations de travail de la décennie précédente commençaient à être menacées par les ordinateurs personnels, plus récents, vendus à faible prix, et aux performances élevées, construits autour d'un processeur de type Intel 80386 ou de l'un de ses descendants. Pour la première fois un hacker pouvait, à titre individuel, acquérir une machine domestique offrant une puissance et une capacité de stockage comparables à celles des mini-ordinateurs disponibles dix ans auparavant, une Unix-ette capable de

proposer un environnement de développement complet et de communiquer sur l'Internet.

Le monde de MS-DOS négligea béatement tout cela. Le nombre de personnes enthousiasmées par la micro (environnements MS-DOS et MacOS) avait rapidement dépassé de quelques ordres de grandeur celui des connectés à la « nation réseau » mais ces passionnés n'ont jamais formé de culture consciente d'elle-même. Le rythme des changements était si élevé que cinquante cultures techniques différentes ont vu le jour pour s'éteindre aussi rapidement que des éphémères, sans jamais atteindre la stabilité nécessaire au développement d'une tradition commune comportant jargon, folklore, et histoires mythiques. En l'absence d'un véritable réseau, comparable à UUCP ou à l'Internet, elles n'ont jamais pu devenir elles-mêmes une nation réseau. L'accès grand public aux services commerciaux en ligne tels que CompuServe et Genie commençait à prendre forme, mais le fait que les systèmes non Unix n'étaient pas livrés avec des outils de développement signifiait qu'il était très difficile d'y compiler du code source. C'est pourquoi il ne s'est développé, dans ces cultures, aucune tradition de hackers travaillant de manière collaborative.

Le courant principal des hackers, (dés)organisés sur l'Internet et qu'on pouvait maintenant clairement assimiler à la culture technique d'Unix, se fichait des services commerciaux. Ils voulaient de meilleurs outils et plus de l'Internet, et des ordinateurs personnels de type PC à architecture 32bits, qui promettaient de mettre tout cela à portée de la main.

Mais qu'en était-il du logiciel ? Les Unix commerciaux demeuraient onéreux (ils coûtaient plusieurs milliers de dollars). Au début des années 1990, plusieurs sociétés ont tenté de vendre les ports d'Unix d'ATT et de BSD sur des machines personnelles de type PC. Elles ont rencontré un succès fort limité, les prix baissaient peu, et (ce qui était le pire) on ne disposait pas du code source du système d'exploitation, qu'on ne pouvait donc pas modifier et redistribuer. Le modèle traditionnel des entreprises de logiciels ne donnait pas aux hackers ce qu'ils attendaient.

La FSF ne leur proposait pas de système. Le développement de Hurd, le noyau libre promis depuis longtemps par RMS aux hackers, s'est embourbé pendant de nombreuses années et n'a commencé à produire un noyau vaguement utilisable qu'en 1996 (alors que la FSF proposait dès 1990 la plupart des autres portions, dont les plus compliquées, d'un système d'exploitation de type Unix).

Pis, au début des années 1990, il devenait limpide que dix années d'efforts de commercialisation des Unix propriétaires se soldaient par un échec. La portabilité d'une plate-forme à l'autre, grande promesse d'Unix, avait cédé le pas aux chamailleries induites par une demi-douzaine de versions propriétaires d'Unix. Les acteurs du monde Unix propriétaire se sont révélés si peu dynamiques, si aveugles, et si inaptes à la mercatique, que la société Microsoft a pu prendre une large portion de leur marché avec son système d'exploitation MS-Windows, pourtant étonnamment inférieur sur le plan technique.

Au début de l'année 1993, un observateur hostile pouvait penser que l'histoire d'Unix était sur le point de se conclure, et qu'avec elle disparaîtrait la bonne fortune de la tribu des hackers. Et on ne manquait pas d'observateurs hostiles dans la presse informatique professionnelle, beaucoup d'entre eux ayant régulièrement prédit la mort imminente d'Unix depuis la fin des années 1970, selon un rituel semestriel.

À l'époque, il était sage de penser que l'ère du techno-héroïsme individuel

avait pris fin, et que l'industrie du logiciel et l'Internet naissant seraient peu à peu dominés par des colosses comme la société Microsoft. La première génération des hackers Unix semblait vieillissante et fatiguée (le groupe de recherche en informatique de Berkeley s'est essoufflé et a perdu son financement en 1994). Le moral était au plus bas.

Heureusement, des gens avaient concocté, à l'insu de la presse professionnelle et même de la plupart des hackers, de quoi produire des développements extrêmement encourageants à la fin de l'année 1993 et en 1994. À terme, ils seraient à l'origine d'un changement de cap qui, s'imposant à toute la culture des hackers, devait la conduire vers des réussites dont ils n'auraient jamais osé rêver.

2.6 Les premiers Unix libres

Un étudiant de l'université d'Helsinki nommé Linus Torvalds a comblé le vide laissé par l'échec du Hurd. En 1991, il a commencé à développer un noyau Unix libre pour les machines de type Intel 80386, en utilisant la boîte à outils de la Fondation du logiciel libre. Ses premières réussites, rapides, ont attiré de nombreux hackers de l'Internet qui l'ont aidé à développer Linux, un système Unix complet, au code source entièrement libre et redistribuable.

Linux ne manquait pas de concurrents. En 1991, année des premières expériences de Linus Torvalds, William et Lynne Jolitz portaient, de manière expérimentale, les sources de l'Unix de BSD sur le 386. La plupart des observateurs qui comparaient la technique proposée par BSD aux rudes premiers efforts de Linus s'attendaient à voir BSD jouer le rôle du système Unix libre le plus important sur PC.

La spécificité la plus importante de Linux n'était pas d'ordre technique mais bien sociologique. Jusqu'au développement de Linux, tout le monde croyait que tout logiciel aussi compliqué qu'un système d'exploitation devait être développé de manière soigneusement coordonnée par un petit groupe de gens étroitement liés. Ce modèle était et demeure représentatif des logiciels commerciaux et des grandes cathédrales libres construites par la Fondation du logiciel libre dans les années 1980 ; c'était aussi le cas des projets FreeBSD/ NetBSD/OpenBSD, qui ont émergé du port originel de 386BSD assuré par les Jolitz.

Linux a évolué de manière complètement différente. Dès le début, ou presque, des hordes de hackers volontaires se sont échinés à le modifier librement, et la coordination ne se faisait que par l'Internet. Ce n'étaient pas des normes rigides ou l'autocratie qui garantissaient la qualité, mais la publication hebdomadaire du logiciel et la collecte des commentaires de centaines d'utilisateurs quelques jours plus tard, créant ainsi une sorte de sélection darwinienne accélérée sur les mutations introduites par les développeurs. À la surprise générale, ce système a très bien fonctionné.

À la fin de l'année 1993, la stabilité et la fiabilité Linux correspondaient à celles de la plupart des Unix commerciaux, et il proposait davantage de logiciels. Il commençait déjà à susciter les ports d'applications logicielles propriétaires. Un effet indirect de ce développement fut de condamner les petits vendeurs d'Unix commerciaux qui, en l'absence de développeurs et de hackers à qui vendre leur produit, se sont écroulés. L'un des rares survivants, BSDI (Berkeley Systems Design, Incorporated), n'a dû son salut et son essor qu'au fait d'offrir le code source

complet de son système Unix à base BSD et parce qu'il entretient d'étroites relations avec la communauté des hackers.

Ces développements passèrent inaperçus à l'époque, même au sein de la communauté des hackers, et complètement inaperçus à l'extérieur. La culture des hackers, défiant les prédictions répétées de sa mort annoncée, reconstitua à sa façon un équivalent du travail assuré par le monde du logiciel commercial. Cette tendance s'affirma peu à peu durant cinq ans.

2.7 La grande explosion du web

La croissance initiale de Linux s'est produite en même temps qu'un autre phénomène : la découverte de l'Internet par le grand public. Le début des années 1990 a aussi vu le début d'une florissante industrie de fourniture d'accès à l'Internet, qui vendait au particulier la possibilité de se connecter pour quelques dollars par mois. Suite à l'invention du World Wide Web, la croissance de l'Internet, déjà rapide, a atteint une allure folle.

En 1994, l'année ou le groupe de développement de l'Unix de Berkeley s'est officiellement dissous, c'est sur diverses versions libres d'Unix (GNU/Linux et les descendants de 386BSD) que la plupart des hackers focalisaient leurs activités. Le système GNU/Linux, distribué commercialement sur des CD-ROM, se vendait comme des petits pains. À la fin de l'année 1995, les sociétés d'informatique les plus importantes vantaient les mérites de leurs logiciels et matériels en matière d'Internet !

À la fin des années 1990, les hackers se sont concentrés sur le développement de Linux et la popularisation de l'Internet. Le World Wide Web a au moins eu pour effet de transformer l'Internet en medium de masse, et de nombreux hackers des années 1980 et du début des années 1990 fondèrent des sociétés de prestations de services liés à l'Internet en vendant ou en offrant aux masses un accès à l'Internet.

Le passage de l'Internet au premier plan a même apporté aux hackers un peu de respectabilité et un petit rôle politique. En 1994 et 1995, l'activisme hacker a saboté la proposition Clipper, qui aurait placé la cryptographie forte sous le contrôle du gouvernement (des États-Unis d'Amérique). En 1996, les hackers ont mobilisé une large coalition pour défaire le mal nommé « Communications Decency Act » (ou CDA, une proposition de loi de contrôle de la décence des communications), et interdire ainsi la censure sur l'Internet.

La victoire sur le CDA nous fait passer d'un registre historique à un registre d'actualité. On entre aussi dans une période où votre historien joue un rôle plus actif que celui d'observateur. Cette narration continue dans « La revanche des hackers ».

> Les gouvernements sont tous, plus ou moins, des coalitions opposées au peuple...et les dirigeants n'ayant pas plus de morale que ceux qu'ils dirigent...on ne peut maintenir le pouvoir d'un gouvernement dans les limites qu'il s'est imposées qu'en lui faisant la démonstration d'une puissance égale à la sienne, le sentiment de tout un peuple.

Benjamin Franklin Bache, dans un éditorial du *Philadelphia Aurora*, 1794.

Chapitre 3

Deux décennies d'Unix Berkeley — De la version AT&T à la version libre

3.1 La genèse

Ken Thompson et Dennis Ritchie ont présenté le premier document concernant Unix à la conférence Principes des systèmes d'exploitation à l'université de Purdue en novembre 1973. Le professeur Bob Fabry, de l'université de Californie à Berkeley, se trouvait dans le public et souhaita d'emblée obtenir une copie du système afin de le tester à Berkeley.

À cette époque, Berkeley ne possédait que de gros systèmes offrant uniquement du traitements par lots (batch processing). La première démarche visa à obtenir un PDP-11/45 afin de faire fonctionner la version4 d'Unix, alors la plus récente. Le département d'informatique de Berkeley et les départements de mathématiques et de statistique s'associèrent pour acheter ensemble une telle machine. En janvier 1974, une cartouche de la Version 4 fut livrée et un étudiant nommé Keith Standiford l'installa.

Bien que Ken Thompson, installé à Purdue, ne fût pas impliqué dans l'installation réalisée à Berkeley ainsi qu'il l'avait été pour la plupart des systèmes fonctionnant à cette époque, son expertise s'est rapidement avérée nécessaire pour déterminer les causes de plusieurs étranges plantages. Étant donné que Berkeley ne disposait que d'un modem acoustique à 300 bit/s dépourvu de système de décrochage automatique, Thompson devait téléphoner à Standiford dans la salle des machines, qui basculait la ligne sur le modem. De cette manière, Thompson déboguait à distance les traces des plantages depuis le New Jersey.

La plupart des plantages étaient provoqués parce que l'interface disque était incapable d'effectuer des déplacements avec chevauchement, contrairement à ce qu'indiquait la documentation. Ce PDP-11/45 a été l'une des premières machines soumises à Thompson pourvue de deux disques connectés à la même interface. La bonne volonté des chercheurs des laboratoires à partager leur travail avec Berkeley a permis l'amélioration rapide du logiciel.

Unix devint rapidement fiable et fonctionnel, mais le regroupement des départements d'informatique, de mathématiques et de statistique se heurta peu après à un problème car les deux derniers départements voulaient utiliser un système DEC RSTS. L'approche retenue après de nombreux débats laissait à chaque département un créneau horaire de huit heures :Unix fonctionnerait pendant huit heures, puis RSTS pendant seize heures. Pour des raisons d'équité, une rotation des tranches horaires fut mise en place. Unix fonctionnait donc de 8 heures à 16 heures un jour, de 16 heures à minuit le lendemain, et de minuit à 8 heure le jour suivant. En dépit de cette étrange approche les étudiants suivant les cours portant sur les systèmes d'exploitation préféraient mener leurs projets sous Unix plutôt qu'utiliser le système de traitement par lots.

Les professeurs Eugene Wong et Michael Stonebraker se sentaient particulièrement limités par l'environnement de traitement par lots. Leur projet de base de données Ingres fut l'un des premiers à adopter l'environnement interactif d'Unix en lieu et place de RSTS. Ils constatèrent vite que la faible durée du temps machine imparti à Unix et la rotation des créneaux d'exploitation du PDP-11/45 étaient intolérables. Ils achetèrent donc, au printemps 1974, un PDP-11/40 fonctionnant avec la nouvelle Version 5. Avec la première distribution d'Ingres à l'automne 1974, ce groupe est devenu le premier du département d'informatique à distribuer son logiciel. Plusieurs centaines de bandes Ingres furent expédiées aux cours des six années qui suivirent, établissant la réputation

de Berkeley dans le secteur de la conception et de la réalisation de systèmes.

Même après que le projet Ingres ai renoncé à exploiter le PDP-11/45, le temps machine était toujours insuffisant pour les étudiants. Pour réduire les effets de cette pénurie, les professeurs Michael Stonebraker et Bob Fabry demandèrent, en juin 1974, deux PDP-11/45 destinés à améliorer l'enseignement délivré par le département d'informatique. Ils obtinrent le budget au début de l'année 1975. DEC annonça à peu près au même moment le modèle 11/70, une machine semblant beaucoup plus performante que la précédente. L'argent destiné aux PDP-11/45 fut groupé pour acheter une machine 11/70, livrée à l'automne 1975. Ken Thompson décida de prendre une année sabbatique en tant que professeur visiteur à l'université de Californie à Berkeley, revenant ainsi sur le lieu où il avait mené ses études à un moment qui coïncidait avec l'arrivée de cette machine. Thompson, avec l'aide de Jeff Schriebman et de Bob Kridle, installa la Version 6 sur la nouvelle machine 11/70.

À l'automne 1975, deux étudiants diplômés inconnus, Bill Joy et Chuck Haley, arrivèrent également. Ils manifestèrent immédiatement de l'intérêt pour ce nouveau système et travaillèrent tout d'abord sur un Pascal que Thompson avait bricolé alors qu'il se trouvait aux alentours de la salle machine du 11/70. Ils étendirent et améliorèrent l'interpréteur au point qu'il devint le langage de programmation système préféré des étudiants en raison de son excellent système de traitement d'erreurs et de sa vitesse de compilation et d'exécution.

Avec le remplacement des consoles texte Model33 par des terminaux de type ADM-3, Joy et Haley commencèrent a souffrir des limites de l'éditeur ed. En travaillant à partir d'un éditeur appelé em, mis à disposition par le professeur George Coulouris de l'université Queen Mary de Londres, ils créèrent un éditeur ligne à ligne nommé ex.

Après le départ de Ken Thompson à la fin de l'été 1976, Joy et Haley commencèrent à explorer avec intérêt le cur du noyau Unix. Sous les yeux observateurs de Schriebman, ils installèrent en premier les correctifs et améliorations fournies sur la cartouche « cinquante modifications » [1] fournie par les laboratoires Bell. Ayant appris à manipuler le code source, ils suggérèrent plusieurs petites améliorations destinées à lutter contre certaines limites du noyau.

3.2 Les premières distributions

Le travail effectué afin d'améliorer la gestion des erreurs assurée par leur environnement Pascal suscita de l'intérêt. Au début de l'année 1977, Joy créa la Berkeley Software Distribution. Cette première version contenait le système Pascal avec dans un sous-répertoire obscur des sources de ce dernier, l'éditeur ex. Pendant l'année suivante, Joy, agissant en fonction de la capacité de distribution du secrétariat, diffusa environ trente copies gratuites du système.

Après l'arrivée de quelques terminaux ADM-3a offrant la possibilité de gérer le curseur, Joy a finalement pu écrire vi, apportant à Berkeley un éditeur plein écran. Il ne sut rapidement plus trop quoi faire. Comme c'est souvent le cas dans les universités aux moyens financiers limités, l'ancien équipement n'est jamais remplacé que progressivement. Plutôt que d'organiser le code pour optimiser la gestion de types de terminaux différents, il décida de l'améliorer grâce à un petit

1. fifty changes.

interpréteur capable de les gérer grâce à la description des caractéristiques du terminal. Ainsi naquit termcap.

Vers le milieu de l'année 1978, la distribution devait être mise à jour. Le système Pascal avait été rendu sensiblement plus robuste grâce aux différents commentaires de ses utilisateurs, une communauté en pleine expansion. Il fonctionnait désormais en deux passes afin que même le PDP-11/34 permette de l'employer. Le résultat de cette mise à jour fut la Second Berkeley Software Distribution, un nom rapidement abrégé par 2BSD. Elle comprenait, outre le système Pascal amélioré, vi et les définitions termcap de plusieurs types de terminaux. À nouveau, Bill Joy créa la distribution, répondit au téléphone et incorpora les différentes remarques d'utilisateurs. Dans l'année qui suivit, près de 75 cartouches furent vendues. Joy embrassa d'autres projets l'année suivante, mais la distribution 2BSD continua de s'étendre. La version finale de cette distribution, 2.11BSD, était un système complet utilisé sur des centaines de PDP-11 situés aux quatre coins du monde.

3.3 Unix VAX

Au début de l'année 1978, le professeur Richard Fateman chercha une machine offrant un espace d'adressage plus important afin de poursuivre son projet Macsyma (commencé sur un PDP-10). La nouvelle machine VAX-11/780 répondait à ses besoins et convenait à son budget. Fateman et treize autres membres de l'université rédigèrent une proposition NSF qu'ils combinèrent avec certains financements départementaux afin d'acheter un VAX.

Le VAX utilisa tout d'abord le système VMS de DEC, mais le département était désormais habitué à l'environnement Unix et souhaitait poursuivre dans cette voie. Donc, peu après l'arrivée de cette nouvelle machine, Fateman obtint un exemplaire du port 32/V d'Unix VAX réalisé par John Reiser et Tom London aux laboratoires Bell.

Le 32/V offrait un environnement Unix Version 7 mais ne tirait pas profit des capacités de gestion de la mémoire virtuelle du matériel VAX. Comme ses prédécesseurs sur le PDP-11, il s'agissait d'un système chargeant en mémoire vive de toutes les zones mémoire d'un processus actif (swap-based system). Pour le groupe Macsyma de Berkeley, le manque de mémoire virtuelle signifiait que l'espace d'adressage d'un processus était limité par la taille de la mémoire physique, d'un Mo sur le nouveau VAX.

Pour résoudre ce problème, Fateman contacta le professeur Domenico Ferrari, qui s'occupait de système à Berkeley, et lui demanda si son groupe pouvait développer un système de mémoire virtuelle pour Unix. Ozalp Babaoglu, l'un des étudiants de Ferrari, se mit à chercher une manière d'implémenter un système de pagination sur le VAX. Sa tâche n'était pas facile car cette machine n'utilisait pas de bit de référence.

Lorsque Babaoglu fut proche de la première implémentation, il contacta Bill Joy pour obtenir quelques explications sur les méandres du noyau Unix. Intrigué par l'approche de Babaoglu, Joy l'aida à intégrer le code sur le 32/V et à le déboguer.

Malheureusement, Berkeley n'avait qu'une seule machine VAX pour le développement système et la production. Donc, pendant les quelques semaines des vacances de Noël, la communauté tolérante des utilisateurs d'Unix se retrouva

de façon aléatoire sous 32/V et sous Virtual VAX/Unix. Fréquemment, leurs programmes fonctionnant sur le premier système se trouvaient arrêtés de façon brutale, et quelques minutes plus tard, les utilisateurs se retrouvaient avec la bannière de connexion du 32/V. En Janvier 1979, la plupart des bogues avaient été corrigées et le 32/V releva dès lors du passé.

Joy perçut que le VAX 32 bits rendrait bientôt le PDP-11 16 bits obsolète et commença à porter la version 2BSD sur le VAX. Pendant que Peter Kessler et moi portions le système Pascal, Joy portait les éditeurs ex et vi, le shell C et la myriade d'autres plus petits programmes provenant de la distribution 2BSD. À la fin de l'année 1979, une distribution complète était prête. Cette distribution incluait le noyau gérant la mémoire virtuelle, les utilitaires 32/V standards et les ajouts provenant de 2BSD. En décembre 1979, Joy diffusa près de cent exemplaires de 3BSD, la première distribution VAX de Berkeley.

32/V fut la dernière version provenant des laboratoires Bell. Les versions ultérieures d'Unix diffusées par ATT, tout d'abord SystemIII et plus tard SystemV, furent gérées par un groupe différent qui mit l'accent sur des diffusions commerciales stables. Avec la commercialisation d'Unix, les chercheurs des laboratoires Bell n'étaient plus capables d'agir comme la « maison propre » de la recherche Unix. Comme la communauté de chercheurs continuait à modifier le système Unix, il s'avéra qu'elle avait besoin d'une organisation capable de produire des versions des travaux de recherche. Berkeley, en raison de son implication dans le développement d'Unix et d'outils compatibles, a rapidement endossé le rôle joué auparavant par les laboratoires Bell.

3.4 La DARPA soutient le projet

Au même moment, dans les bureaux dirigeants de la DARPA (agence pour les projets de recherche avancée de défense), certaines discussions en cours eurent un effet majeur sur le travail effectué à Berkeley. L'un des tous premiers succès de la DARPA avait été de mettre en place un réseau d'ordinateurs inter-connectant tous les centres de recherche majeurs des États-Unis. À cette époque, la DARPA découvrait qu'un grand nombre d'ordinateurs exploités dans ces centres vieillissaient, et devaient être remplacés. Les coûts de l'adaptation les logiciels de recherche à ces nouvelles machines dépasseraient alors les autres. De plus, un grand nombre de sites ne pouvaient partager leurs programmes en raison de la diversité des matériels et des systèmes d'exploitation.

La variété des ressources informatiques requises pour ces groupes de recherche interdisait de ne sélectionner qu'un constructeur, et la DARPA ne souhaitait tout livrer ainsi à une seule entité commerciale. Il fut donc décidé d'assurer la standardisation par le système d'exploitation. Après bon nombre de discussions, Unix fut sélectionné car il avait démontré sa portabilité.

À l'automne 1979, Bob Fabry répondit aux souhaits de la DARPA en mettant en valeur Unix dans un document suggérant que Berkeley développe une version améliorée de 3BSD pour la communauté DARPA. Fabry en porta une copie à une réunion de sous-traitants de la DARPA uvrant dans le domaine du traitement d'image, du circuit intégré, à laquelle assistaient aussi des représentants de Bolt, Beranek et Newman, les développeurs d'ARPAnet. Quelques réserves portaient sur la capacité de Berkeley à produire un système d'exploitation utilisable. Toutefois, la diffusion de 3BSD en décembre 1979 dissipa la

plupart des doutes.

Grâce à l'amélioration ininterrompue de la réputation de la version 3BSD, Bob Fabry fut capable d'obtenir un contrat de 18 mois avec la DARPA, commençant en avril 1980. Ce contrat consistait à ajouter certaines fonctionnalités nécessaires pour les contracteurs de la DARPA. Sous les auspices de ce contrat, Bob Fabry mit en place une organisation baptisée Groupe de Recherche en Systèmes Informatiques (le « Computer Systems Research Group », ou CSRG). Il embaucha immédiatement Laura Tong pour qu'elle s'occupe de l'administration du projet. Fabry chercha ensuite un chef de projet de développement logiciel. Il supposa qu'étant donné que Joy avait tout juste passé son examen d'entrée en thèse, il choisirait de se concentrer sur ces études plutôt que de prendre ce poste. Mais Joy avait d'autres objectifs. Une nuit du début du mois de mars, il téléphona à Fabry pour lui indiquer qu'il souhaitait prendre en charge les futurs développements d'Unix. Quoique surpris par cette offre, Fabry accepta très rapidement.

Le projet débuta à vive allure. Tong mit en place un système de distribution pouvant supporter un volume de commandes supérieur à celui que permettaient les procédures en place. Fabry coordonna ses efforts avec ceux de Bob Guffy (ATT) et des avocats de l'université de Californie pour diffuser de façon officielle Unix suivant des termes acceptables par tous. Joy incorpora le contrôle de tâches de Jim Kulp et il ajouta un système d'auto-réamorçage, un système de fichiers utilisant des blocs d'1 Ko et la gestion de la dernière machine VAX, le VAX 11/750. En octobre 1980, une distribution proche de la perfection incluant le compilateur Pascal, l'interpréteur Lisp de Franz ainsi qu'un système amélioré de gestion de courrier électronique était diffusé, sous le nom de 4BSD. Environ 150 exemplaires de cette distribution furent vendues au cours de ses neuf mois de vie. Le système de licence mis en place concernait les organisations plutôt que les machines, de sorte que la distribution fonctionna donc sur environ 500 machines.

Des critiques furent formulées car la diffusion de la distribution et l'activité de l'Université Berkeley augmentaient. David Kashtan, de l'Institut de recherche de Stanford, écrivit un article décrivant les résultats de tests de performances menés à la fois sous VMS et sous l'Unix Berkeley [2]. Ces tests accordaient à la version d'Unix destinée au VAX des performances fort limitées. Joy, délaissant temporairement ses projets, évalua systématiquement les performances du noyau et rédigea en quelques semaines un article montrant que les tests de performances sous Unix pouvaient s'avérer tout aussi favorables à Unix qu'à VMS.

La diffusion de 4.1BSD commença en juin 1981. Ce système remplaçait 4BSD car résultait de l'ajout du code d'auto-configuration écrit par Robert Elz et des modifications effectuées lors de la campagne d'évaluation. Il vécut environ deux ans, et près de 400 distributions furent vendues. Cette version aurait dû s'appeler 5BSD mais ATT refusa car redoutait la confusion entre leur Unix commercial, SystemV, et une version de l'Unix Berkeley, 5BSD. Pour résoudre ce problème, Berkeley accepta de changer le système de numérotation pour les futures versions de manière à conserver 4BSD, et à n'incrémenter que le numéro mineur.

2. ces deux universités entretiennent une féroce rivalité bien que quelques dizaines de kilomètres seulement les séparent

3.5 4.2BSD

Avec la diffusion de 4.1BSD, la plupart des remarques désagréables concernant les performances disparurent. La DARPA, suffisamment satisfaite des résultats du premier contrat, signa avec Berkeley un nouveau contrat de deux ans, au budget presque cinq fois supérieur à celui du premier contrat. La moitié de cette somme servit au projet Unix, l'autre à d'autres projets de recherche au département d'informatique. Il s'agissait d'effectuer des travaux majeurs afin que la communauté des chercheurs de la DARPA puissent travailler plus facilement.

Des objectifs correspondants aux besoins de cette communauté furent fixés, puis le mode de définition des modifications à apporter au système. En particulier, la nouvelle version devait offrir un système de fichiers plus rapide donc adapté aux performances des disques disponibles, la gestion de processus sur des espaces d'adressage de plusieurs giga-octets, un système de communication inter-processus souple grâce auquel les chercheurs s'intéresseraient à la distribution des travaux, et une gestion réseau pour que les machines utilisant le nouveau système puissent facilement participer à l'ARPAnet.

Pour faciliter la définition du nouveau système, Duane Adams, un contracteur de Berkeley rédacteur au DARPA, constitua un groupe connu sous le nom de « comité de pilotage » chargé de guider le travail de conception et de s'assurer de la prise en compte des besoins de la communauté des chercheurs. Ce comité se réunit deux fois par an entre avril 1981 et juin 1983. Il était constitué de Bob Fabry, Bill Joy et Sam Leffler de l'université de Californie à Berkeley ; Alan Nemeth et Rob Gurwitz de Bolt, Beranek et Newman ; Dennis Ritchie des laboratoires Bell ; Keith Lantz de l'université Stanford ; Rick Rashid de l'université Carnegie-Mellon ; Bert Halstead du MIT (Massachusetts Institute of Technology) ; Dan Lynch de l'Institut des sciences de l'information ; Duane Adams et Bob Baker de la DARPA et Jerry Popek de l'université de Californie de Los Angeles. Au début de 1984, ces réunions furent supplantées par des ateliers ouverts à de nombreuses autres personnes.

Un document préliminaire décrivant les fonctionnalités a inclure dans le nouveau système circula au sein du comité de direction et fut distribué à quelques personnes externes à Berkeley en juillet 1981, engendrant de longs débats. Pendant l'été 1981, je fus impliqué au CSRG et je pris la charge du nouveau système de fichiers. Pendant cette période, Joy se concentra sur l'implémentation d'une version prototype du système de communication inter-processus. À l'automne 1981, Sam Leffler devint membre à plein temps du CSRG pour travailler avec Bill Joy.

Lorsque Rob Gurwitz diffusa une version préliminaire de l'implémentation des protocoles TCP/IP à Berkeley, Joy l'intégra dans le système et évalua ses performances. Au cours de cette tâche, il devint évident à Joy et à Leffler que le nouveau système devait gérer d'autres protocoles que ceux du DARPA. Ils réformèrent l'architecture interne du logiciel, raffinant les interfaces de telle manière que plusieurs protocoles réseau puissent être utilisés en même temps.

Avec la restructuration interne du noyau effectué et les protocoles TCP/IP intégrés avec le prototype des IPC, plusieurs applications assez simples furent créées pour fournir aux utilisateurs locaux un accès à des ressources distantes. Les programmes rcp, rsh, rlogin et rwho étaient supposés être en fin de compte des outils temporaires à remplacer par des versions aux fonctionnalités raison-

nables (d'où le préfixe « r » qui les distinguait). Ce système, appelé 4.1a, fut distribué en avril 1982 pour une utilisation locale : il n'avait jamais été destiné à une large diffusion, même si des copies faites en contrebande proliférèrent sur les sites impatients de recevoir la version 4.2.

Le système 4.1a était obsolète longtemps avant son achèvement. Toutefois, les commentaires de ses utilisateurs fournirent des informations très importantes. Elles furent utilisées pour créer un nouveau document pour le système appelé « 4.2BSD System Manual ». Ce document diffusé en février 1982 contenait une description concise mais complète de 4.2BSD, des interfaces utilisateurs aux fonctionnalités du système.

Tout en développant la version 4.1a, je terminais l'implémentation du nouveau système de fichiers et en juin 1982, il était totalement intégré dans le noyau 4.1a. Le système résultant fut baptisé 4.1b et ne fonctionnait que sur certaines machines de développement à Berkeley. Joy sentit qu'avec les modifications particulièrement dangereuses apportées au système, il était préférable d'interdire toute distribution, même locale, en particulier depuis qu'il était nécessaire que chacun des systèmes de fichiers soit archivé et restauré pour le convertir de la version 4.1a vers la version 4.1b. Sitôt le code du système de fichiers stabilisé, Leffler ajouta le nouveau système de fichiers aux différents appels système, pendant que Joy travaillait sur une modification du système de communication inter-processus.

À la fin du printemps 1982, Joy annonça qu'il rejoignait Sun Microsystems. Pendant l'été, il partagea son temps entre Sun et Berkeley, acheva durant la majeure partie de son temps le code de communication inter-processus et réorganisa le code source du noyau Unix pour isoler les parties dépendantes de la machine. Avec le départ de Joy, Leffler prit la responsabilité du projet. Certaines dates limites avaient déjà été fixées et les diffusions promises à la communauté de la DARPA pour le printemps 1983. En raison des contraintes de temps, le travail restant à effectuer pour terminer cette version fut évalué et des priorités fixées. En particulier, les améliorations de la gestion de la mémoire virtuelle et les parties les plus sophistiquées de la conception de la communication inter-processus virent leur priorité réduite (et plus tard totalement annulée). De plus, après un travail d'implémentation de plus d'un an et une attente de la communauté Unix de plus en plus pressante, il fut décidé qu'une version intermédiaire serait diffusée pour ménager la patience des intéressés, avant l'avènement d'une version finale. Ce système, appelé 4.1c, fut distribué en avril 1983. Bon nombre de constructeurs utilisèrent cette version pour préparer le portage de la version 4.2 sur leur matériel. Pauline Schwartz fut embauchée pour la distribution à partir de la version 4.1c.

En juin 1983, Bob Fabry laissa le contrôle administratif du CSRG aux professeurs Domenico Ferrari et Sysan Graham pour commencer une année sabbatique loin de l'activité frénétique des quatre dernières années. Leffler continua à compléter le système, à implémenter le nouveau système de signaux et la gestion du réseau, à réécrire le système indépendant de gestion des entrées/ sorties pour simplifier le système d'installation, à intégrer la gestion des quotas venant de Robert Elz, à mettre à jour toute la documentation et à chercher les bogues de la version 4.1c. En août 1983, le système fut diffusé sous le nom de 4.2BSD.

Lorsque Leffler quitta Berkeley pour Lucasfilm à la suite de la mise en place de la version 4.2, il fut remplacé par Mike Karels. L'expérience précédente de Karels avec le programme de distribution de 2.1BSD sur PDP-11 lui conférait le

profil idéal pour ce nouveau travail. Après avoir terminé ma thèse en décembre 1984, je rejoignis Mike Karels à plein temps au CSRG.

4.2BSD connut une popularité impressionnante. En 18 mois, plus de 1 000 licences site furent diffusées. Le nombre d'exemplaires de 4.2BSD vendus dépassa celui de toutes les autres distributions de logiciels de Berkeley confondues. La plupart des vendeurs d'Unix vendaient plutôt le système 4.2BSD que le système commercial SystemV d'ATT car ce dernier n'avait ni gestion réseau, ni le système de fichiers rapide de Berkeley. La version BSD ne détint cette position dominante dans le secteur commercial que durant quelques années, avant de retourner à ses racines universitaires. À mesure que le code réseau et les autres améliorations de 4.2BSD furent intégrés dans la version SystemV, les constructeurs l'adoptèrent à nouveau. Toutefois, les développements ultérieurs de BSD continuèrent à être intégrés dans SystemV.

3.6 4.3BSD

Les critiques portant sur 4.2BSD naquirent peu après sa publication. La plupart portaient sur la lenteur du système. Le problème était, sans grande surprise, que les nouvelles fonctionnalités avaient été ajoutées sans vraiment être optimisées et que trop de structures de données du noyau n'étaient pas vraiment adaptées à leur nouvelle utilisation. J'ai consacré la première année de mon activité sur ce projet à évaluer et fignoler le système, avec Karels.

Après deux années de travail d'optimisation du système et d'amélioration du code réseau nous avons annoncé, lors de la conférence Usenix de juin 1985, que nous espérions diffuser 4.3BSD pendant l'été. Toutefois, ce projet de diffusion fut rapidement stoppé par les gens de BBN. Ils mirent en évidence que nous n'avions jamais mis à jour 4.2BSD avec la version finale de leur code réseau. Au contraire, nous utilisions plutôt le prototype initial, très modifié, qu'ils nous avaient donné de nombreuses années auparavant. Ils se plaignirent à la DARPA que Berkeley devait implémenter l'interface alors que BBN était supposé implémenter le protocole et que Berkeley devait par conséquent remplacer le code TCP/IP de 4.3BSD par l'implémentation BBN.

Mike Karels reçut le code de BBN et effectua une évaluation du travail effectué depuis que le prototype avait été livré à Berkeley. Il décida que la meilleure chose à faire était d'incorporer les bonnes idées du code BBN dans le code Berkeley plutôt que de remplacer ce dernier, bien testé et contenant d'innombrables améliorations issues de la grande diffusion qu'avait connue 4.2BSD. Toutefois, sous la forme d'un compromis, il offrit d'inclure les deux implémentations dans la distribution 4.3BSD et de laisser le choix à l'utilisateur.

Après avoir reçu la proposition de Mike Karels, la DARPA décida que diffuser deux codes de base provoquerait obligatoirement des problèmes d'interopérabilité, et décida qu'une seule implémentation devait être diffusée. La DARPA donna ce code à Mike Muuse du laboratoire de recherche en balistique, considéré par toutes les parties comme un tiers indépendant, afin de le laisser isoler le code à utiliser. Après un mois d'évaluation, la conclusion du rapport fut que le code Berkeley était plus efficace mais que le code BBN gérait mieux les problèmes de congestion. Le point de rupture était que le code Berkeley répondait à tous les tests alors que le code BBN provoquait des « paniques » (erreurs fatales) dans certaines conditions de stress. La décision finale de la DARPA fut que 4.3BSD

conserverait le code Berkeley de base.

Le système 4.3BSD particulièrement finalisé fut diffusé en juin 1986. Comme on s'y attendait, il fit taire la plupart des critiques concernant les performances, tout comme la version 4.1BSD avait éliminé les points douteux de 4BSD. Bien que la majorité des constructeurs avaient commencé à rebasculer vers SystemV, de grosses parties de 4.3BSD se trouvaient dans leurs systèmes, en particulier pour le sous-système réseau.

En octobre 1986, Keith Bostic rejoignit le CSRG. L'une des conditions associées à son embauche était qu'il soit autorisé à terminer un projet de portage de 4.3BSD sur le PDP-11, commencé durant son précédent emploi. Karels et l'auteur tenaient pour impossible de porter un système jaugeant 250 Ko sur un VAX de sorte qu'il tienne dans l'espace d'adressage de 64 Ko du PDP-11, et nous avons donc accepté que Bostic termine ses tentatives. À notre grand étonnement le portage fonctionna, en utilisant un ensemble complexe dde recouvrements (overlays) et d'états du processeur auxiliaire. Le résultat fut la version 2.11BSD réalisée par Casey Leedom et Bostic, qui est toujours utilisée sur certains des derniers PDP-11 encore en production en 1998.

Dans le même temps, il devint de plus en plus évident que l'architecture VAX atteignait ses limites et qu'il était temps de considérer d'autres machines pour y faire fonctionner BSD. Computer Console Inc. proposait une nouvelle architecture prometteuse, appelée Power 6/32. Elle s'éteignit malheureusement lorsque la société décida de changer de stratégie. Toutefois, ils avaient déjà fourni au CSRG plusieurs machines grâce auxquelles nous avons terminé le travail, commencé par Bill Joy, qui consistait à isoler dans le noyau BSD les parties dépendantes de la machine de celles qui ne l'étaient pas. La version 4.3BSD-Tahoe naquit ainsi en juin 1988. Le nom « Tahoe » provient du nom de projet utilisé par Computer Consoles Incorporated, pour la machine qu'ils pensaient diffuser avec le Power 6/32. Bien que la durée de vie de la machine Power 6/32 fut courte, les modifications opérées dans le noyau afin d'isoler les parties dépendantes de la machine s'avérèrent extrêmement utiles lors du portage de BSD sur bon nombre d'autres architectures.

3.7 Networking Release 1

Jusqu'à la diffusion de la version 4.3BSD-Tahoe, tous les utilisateurs de BSD devaient, afin de disposer des sources, obtenir une licence auprès d'ATT. En effet, les systèmes BSD ne furent jamais diffusés par Berkeley sous forme binaire. Les distributions contenaient toujours le source complet de chaque partie du système. L'histoire du système Unix et de BSD en particulier a montré l'intérêt de rendre les sources disponibles aux utilisateurs. Au lieu d'utiliser le système de façon passive, ils travaillèrent activement à la correction des erreurs, à améliorer les performances et même à ajouter de nouvelles fonctionnalités.

Avec l'augmentation du coût des licences ATT, les constructeurs qui souhaitaient avoir des produits réseau indépendants utilisant TCP/IP pour le marché PC utilisant le code BSD jugèrent que les tarifs des binaires devenaient prohibitifs. Ils demandèrent donc à Berkeley de séparer le code réseau et les utilitaires associés du reste, et de les leur fournir de façon indépendante de la licence ATT. Le code TCP/IP n'existait clairement pas dans la version 32/V : il fut entièrement développé par Berkeley et ses contributeurs. Le code réseau provenant de

BSD ainsi que les programmes utilitaires associés furent diffusés en Juin 1989 sous le nom de Networking Release 1, le premier code librement redistribuable de Berkeley.

Les termes de la licence étaient peu contraignants. Toute personne pouvait diffuser le code modifié ou non sous forme de code source ou de binaire sans avoir à reverser des droits à Berkeley si les textes de Copyright dans les fichiers code source étaient conservés tels quels et que leur documentation révélait l'utilisation de code provenant de l'université de Californie et de ses contributeurs. Bien que Berkeley fit payer 1 000 dollars pour obtenir une cartouche, n'importe qui était libre d'en obtenir une copie de quelqu'un qui en avait déjà une. En réalité, plusieurs gros sites laissèrent la distribution, peu après sa diffusion, sur leurs sites ftp anonymes. Étant donné qu'elle était facilement accessible, le CSRG fut heureux de voir plusieurs centaines d'organisations en acheter des exemplaires. L'argent aida à financer de nouveaux développements.

3.8 4.3BSD-Reno

Dans le même temps, le développement se poursuivit au niveau du noyau. Le système de mémoire virtuelle dont les interfaces avaient été décrites au départ dans le document de l'architecture de 4.2BSD fut mis à l'ordre du jour. Comme ce fut souvent le cas au CSRG, nous avons toujours essayé de trouver du code existant pour l'intégrer plutôt que d'avoir à réécrire quelque chose depuis zéro. Donc, au lieu de concevoir un nouveau système de gestion de mémoire virtuelle, nous avons recherché des solutions existantes. Notre premier choix fut le système de mémoire virtuelle implanté dans le système SunOS de Sun Microsystems. Bien que certaines discussions eurent lieu à propos de contributions de Sun au code de Berkeley, rien n'aboutit. Nous avons alors considéré une autre possibilité, qui était d'incorporer le système de mémoire virtuelle provenant du système d'exploitation Mach créé à l'université de Carnegie-Mellon. Mike Hibler de l'université d'Utah regroupa les apports techniques de Mach sous l'interface décrite par le manuel de l'architecture de 4.2BSD (qui était également celle de SunOS).

L'autre ajout majeur au système fut la version compatible Sun du système de fichiers réseau, NFS. Encore une fois, le CSRG évita de tout réécrire en incorporant une implémentation réalisée par Rick Macklem de l'université canadienne Geulph.

Bien que nous n'avions pas toutes les fonctionnalités nécessaires à la diffusion de 4.4BSD, le CSRG décida de publier une version intermédiaire pour obtenir plus de commentaires et de réflexions concernant les deux ajouts majeurs au système. Cette version intermédiaire fut appelée 4.3BSD-Reno et fut diffusée début 1990. On a choisi le nom d'une ville qui attire de nombreux flambeurs, Reno (Névada), pour rappeler de façon indirecte aux utilisateurs qu'il est toujours un peu risqué d'utiliser une version intermédiaire[3].

3. les casinos, les jeux de hasard, ainsi que bien d'autres choses, sont interdites en Californie. La ville de Reno située à côté du lac Tahoe se trouve au Névada, à environ quatre heures de route de San Francisco. De nombreux californiens s'y rendent fréquemment pour y respirer un air de liberté).

3.9 Networking Release 2

Lors d'une réunion hebdomadaire du CSRG, Keith Bostic aborda le sujet de la popularité de la version réseau librement diffusée et demanda s'il était possible d'effectuer une version améliorée qui puisse inclure plus de code BSD. Mike Karels et moi dûmes lui signaler que diffuser de larges parties du système était une tâche demandant un travail considérable, mais que s'il pouvait se débrouiller pour réimplémenter les quelques centaines d'utilitaires ainsi que la bibliothèque C nous pourrions bien nous occuper du noyau de la même manière. Nous pensions en nous-mêmes que cela mettrait fin à la discussion.

Comme si de rien n'était, Bostic devint le pionnier en matière de fédération de développeurs assurée par le réseau Internet. Il demanda aux utilisateurs de réécrire les utilitaires Unix en ne repartant que des descriptions publiées. Leur unique compensation serait d'avoir leur nom inscrit en tant que contributeur Berkeley à côté du nom de leur uvre. Les contributions commencèrent lentement et portaient surtout sur des utilitaires triviaux. Mais la liste continua à augmenter au fil des ajouts et des appels à contribution que lançait Bostic lors d'événements publics tels qu'Usenix. La liste compta dès lors plus de cent éléments, et en moins de 18 mois, pratiquement tous les utilitaires et bibliothèques importants furent réécrits.

Bostic entra alors fièrement dans nos bureaux, la liste à la main, voulant savoir où nous en étions avec le noyau. Résignés à effectuer le travail, Karels, Bostic et moi passèrent les quelques mois qui suivirent à parcourir toute la distribution, fichier par fichier, supprimant le code qui provenait de la version 32/V initiale. Lorsque les choses s'éclaircirent, nous découvrîmes alors qu'ils n'y avait plus que six fichiers réseau toujours contaminés et qui ne seraient pas faciles à réécrire. Au lieu de réécrire ces fichiers pour effectuer une diffusion de tout le système, nous décidâmes de diffuser ce que nous avions. Toutefois, il nous fallait la permission des individus placés un peu plus haut dans l'administration de l'université pour diffuser cette version étendue. Après de nombreux débats et vérification de notre méthode pour déterminer la propriété du code, nous reçûmes le feu vert pour effectuer la diffusion.

Notre première idée était d'utiliser un tout nouveau nom pour notre seconde diffusion librement diffusable. Toutefois, il aurait était nécessaire de réécrire une nouvelle licence et de la faire approuver par les avocats de l'université, ce qui était une pure perte de ressources et de temps. Nous décidâmes alors d'appeler cette version, Networking Release2 car nous ne pouvions qu'effectuer une modification de la licence déjà acceptée de Networking Release 1. La diffusion de cette seconde version librement redistribuable particulièrement augmentée débuta en juin 1991. Les termes et le coût étaient les mêmes que pour la première version. Comme la première fois, des centaines de particuliers et d'organisations payèrent 1 000 $ pour obtenir directement la distribution Berkeley.

Combler le vide de la seconde version pour obtenir un système totalement opérationnel ne prit pas longtemps. Moins de six mois après la diffusion, Bill Jolitz écrivit les remplaçants des six fichiers. Il diffusa rapidement un système compilé et amorçable pour l'architecture PC qu'il appela 386/BSD. La distribution 386/BSD de Jolitz s'effectua presque totalement via le réseau Internet. Il configura tout simplement un site FTP anonyme et laissa n'importe qui télécharger la distribution gratuitement. Dans les semaines qui suivirent, il eut

énormément d'adeptes.

Malheureusement, les exigences du travail à plein temps de Jolitz ne lui permettaient pas de gérer le flot de corrections d'erreurs et d'améliorations destinées à 386/BSD. Quelques mois après sa diffusion, un groupe d'utilisateurs motivés constitua le groupe NetBSD pour l'aider à maintenir et à améliorer le système. Leurs diffusions sont désormais connues sous le nom de distribution NetBSD. Ce groupe décida de mettre l'accent sur le portage sur autant de plates-formes que possible et continua le développement orienté recherche tel qu'il était pratiqué au CSRG. Jusqu'en 1998, leurs distributions n'étaient effectuées qu'en utilisant l'Internet : aucun support physique n'était disponible. Ce groupe continue à cibler les utilisateurs avertis. Le site http://www.netbsd.org offre davantage d'informations à ce propos.

Le groupe FreeBSD fut créé peu de mois après la création du groupe NetBSD. L'objectif de ce groupe est de ne prendre en charge que les architectures PC et de plaire au plus grand nombre, même aux utilisateurs peu soucieux des aspects techniques. Il ressemble en cela à Linux. Ce groupe créa des scripts d'installation élaborés et commença à vendre ses CD-ROM à faible prix. L'association de la facilité d'installation avec une large promotion sur le réseau et quelques démonstrations grand public tels qu'au Comdex permirent à ce groupe de prendre très rapidement de l'importance. FreeBSD est sûrement aujourd'hui la base la plus installée des systèmes dérivés de la Release2.

FreeBSD a également profité de la vague de la popularité Linux grâce à l'ajout d'un mode d'émulation Linux qui lui permet de faire fonctionner les exécutables Linux. Les utilisateurs de FreeBSD, grâce à elle, utilisent si nécessaire les applications destinées à Linux, dont le nombre augmente sans cesse, tout en bénéficiant de la robustesse, de la fiabilité et des performances du système FreeBSD. Le groupe a récemment ouvert un centre en ligne (http://www.FreeBSDmall.com), qui regroupe une grosse partie de la communauté FreeBSD, incluant les services de cabinets de conseil, des produits dérivés, des livres et certains articles.

Au milieu des années 1990, OpenBSD se sépara de NetBSD. Leur objectif technique concerne particulièrement la sécurité du système. Leur objectif commercial était de rendre le système plus facile à utiliser et plus accessible. Ils commencèrent à produire et à vendre des CD-ROM avec un bon nombre d'idées concernant la simplicité d'installation provenant de FreeBSD. Vous pouvez obtenir de plus amples informations sur ce projet en consultant le site http://www.OpenBSD.org.

3.10 La saga judiciaire

En plus des groupes organisés pour créer des systèmes libres créés à partir de la distribution Networking Release2, une société Berkeley Software Design, Incorporated (BSDI), fut créée pour développer et distribuer une version du code dont les utilisateurs bénéficieraient d'un service d'assistance commercial (voir le site http://www.bsdi.com). Comme les autres groupes, ils commencèrent par ajouter les six fichiers manquant que Bill Jolitz avait écrits pour sa version 386/BSD. BSDI commença en janvier 1992 à vendre son système, qui comprenait à la fois le code source et les binaires, pour la somme de 995 $. Il publièrent des publicités mettant en évidence que leur produit permettait 99 $ d'économies sur le prix de SystemV, et intégrait le coût du code source. Les lecteurs intéressés

pouvaient appeler le numéro 1-800-ITS-Unix [4].

Ils reçurent, peu après le début de leur campagne de promotion, une lettre expédiée par USL (Unix System Laboratories), une entreprise née de l'éclatement d'ATT dont les activités étaient centrées autour d'Unix et de ses outils. Cette lettre demandait que BSDI arrête de promouvoir ses produits en tant qu'Unix et, en particulier, qu'il cesse d'utiliser le numéro de téléphone qui pourrait tromper son utilisateur. Ce dernier fut rapidement supprimé et les publicités modifiées pour expliquer que le produit n'était pas Unix, mais USL n'était toujours pas satisfait et porta plainte contre BSDI afin de lui interdire de vendre. La plainte indiquait que le produit BSDI contenait du code USL propriétaire et que BSDI diffusait les secrets de fabrication d'USL. USL chercha à obtenir une injonction de blocage immédiat des ventes de BSDI en prétendant qu'ils subiraient sinon des dommages irréparables en raison de la perte de leurs secrets de fabrication.

Lors de la première audience concernant la demande d'injonction, BSDI soutint qu'ils ne faisaient qu'utiliser le code source librement distribué par l'université de Californie et six fichiers ajoutés. Ils acceptaient de discuter du contenu de ces six fichiers, mais ne pensaient pas qu'ils devaient être tenus pour responsables des fichiers diffusés par l'université de Californie. Le juge reconnut les arguments de BSDI et signifia à USL qu'ils devraient reformuler leur plainte de sorte qu'elle ne concerne que les six fichiers ou bien l'abandonner. Reconnaissant qu'ils auraient en ce cas du mal à plaider, USL décida de porter plainte à la fois contre BSDI et contre l'université de Californie. Comme la première fois, USL demanda une injonction bloquant la commercialisation de la Networking Version 2 de l'université de Californie et sur les produits diffusés par BSDI.

Avec l'injonction imminente dont nous n'avions entendu parler que quelques semaines auparavant, la préparation devint particulièrement sérieuse. Tous les membres du CSRG durent faire des dépositions tout comme toute personne employée à BSDI. Les dossiers, les contre-dossiers et les contre-contre dossiers commencèrent leurs allées-venues entre les avocats. Keith Bostic, comme moi, a été contraint d'écrire plusieurs centaines de pages qui trouvèrent place dans plusieurs dossiers.

En décembre 1992, Dickinson R. Debevoise, un juge de district (United States District Judge) du New Jersey, écouta les arguments de l'injonction. Bien que les juges prononcent généralement le résultat d'une requête d'injonction immédiatement, il décida de prendre un certain temps de réflexion. Environ six semaines plus tard, un vendredi, il publia son jugement d'environ quarante pages dans lequel il rejetait l'injonction et repoussait toutes les plaintes sauf deux. Les deux points restants concernaient les nouvelles notices de copyright et la possibilité de pertes consécutives à la dissémination d'un secret industriel. Il suggéra également que cette affaire soit plutôt traitée dans une cour d'état avant de l'être dans une cour fédérale. L'université de Californie appliqua la suggestion et se précipita à la cour d'état de Californie le lundi matin qui suivit avec un dossier contre USL. En déposant en premier en Californie, l'université a établi le lieu de n'importe quelle future action en justice. D'après la constitution, les plaintes ne peuvent être déposées que dans un seul état pour éviter qu'un plaignant aux poches profondes ne puisse saigner une partie adverse en intentant un procès dans chaque état. Le résultat de cette action est que si USL voulait agir contre l'université dans n'importe quelle cour d'état, il était contraint de le faire en

4. numéro vert aux États-Unis/Canada.

Californie plutôt que dans son état, le New Jersey [5].

La plainte déposée par l'université reposait sur le fait qu'USL n'avait pas respecté l'obligation de citer l'Université de Californie dans le code BSD employé dans SystemV, malgré les termes de la licence signée. Si la plainte déposée était considérée recevable, l'université demandait à USL d'être forcé à réimprimer toute leur documentation après ajout des mentions appropriées, d'informer tous les utilisateurs référencés de leur oubli et d'effectuer des publications dans des revues telles que The Wall Street Journal et le magazine Fortune pour que le monde des affaires soit au courant de cette omission.

Peu après le dépôt de cette plainte, USL fut achetée à ATT par Novell. Le directeur de Novell, Ray Noorda, indiqua publiquement qu'il préférait de loin se battre sur le marché plutôt qu'en justice. Pendant l'été 1993, des discussions en vue de résoudre cette affaire débutèrent. Malheureusement, les deux parties s'étaient tellement investies dans ces actions qu'elles évoluaient peu. Avec quelques petits coups de pouces de Ray Noorda du côté d'USL, un grand nombre de points bloquants furent supprimés et cela mena à un arrangement conclu en janvier 1994. Le résultat était que trois fichiers furent supprimés des 18 000 fichiers constituant Networking Release2. L'université accepta d'ajouter les copyrights d'USL dans environ 70 fichiers, tout en continuant à les distribuer librement.

3.11 4.4BSD

La nouvelle version ainsi consacrée fut nommée 4.4BSD-Lite [6] et diffusée en juin 1994 sous une licence identique à celle de Networking. Plus précisément, ses termes autorisaient la distribution gratuite du code source et des binaires à l'unique condition que les copyrights de l'université restent intacts et que l'université soit citée lorsque d'autres utilisent ce code. De façon simultanée, le système complet fut diffusé sous le nom de 4.4BSD-Encumbered [7], qui obligeait toujours à l'utilisateur à détenir la licence source d'USL.

L'accord à l'amiable stipulait également qu'USL ne pourrait poursuivre une quelconque organisation utilisant 4.4BSD-Lite en tant que base de son système. Tous les groupes BSD actifs (BSDI, NetBSD et FreeBSD) durent reprendre leur code de base en se fondant sur celui de 4.4BSD-Lite dans lequel ils intègrent leurs améliorations. Bien que cette réintégration ait un moment freiné le développement des différents systèmes BSD, cela fut une véritable bénédiction camouflée car cela força les groupes divergents à se synchroniser avec les trois ans de développements qui avaient eu lieu au CSRG depuis la diffusion de Networking Release2.

3.12 4.4BSD-Lite, Release 2

L'argent reçu des versions 4.4BSD-Encumbered et 4.4BSD-Lite fut utilisé pour alimenter l'effort d'intégration des corrections de bogues et d'amélioration.

5. Les États-Unis se distinguent par un système juridique d'une complexité effarante. Ce double système juridique ne simplifie pas les choses.

6. « lit » signifie « éclairé », et « light » est la traduction d'« allégé ».

7. la 4.4BSD-Encombrée.

Ces modifications continuèrent pendant deux ans jusqu'à ce que la fréquence des rapports de bogues et des améliorations soit au plus bas. L'ensemble des dernières modifications fut diffusé sous la forme de 4.4BSD-Lite, Release2 en juin 1995. Les autres systèmes dérivé de ce code source bénéficièrent de la plupart de ces modifications.

Après la diffusion de cette dernière version, le CSRG fut dissous. Après environ deux décennies passées à diriger le vaisseau BSD, nous avions senti qu'il était temps que d'autres, aux idées nouvelles et à l'enthousiasme sans limite, prennent le relais. Bien qu'il ait semblé préférable d'avoir une seule autorité de centralisation supervisant l'ensemble du développement du système, l'idée d'avoir plusieurs groupes aux chartes différentes assure que différentes approches seront adoptées. Comme le système est diffusé sous la forme d'un code source, les meilleures idées peuvent être prises d'un système et intégrées dans les autres. Si un groupe devient particulièrement efficace, il est possible que son système domine.

Aujourd'hui, le mouvement Open Source gagne de jour en jour attention et respect. Bien que le système Linux soit probablement le plus connu, près de la moitié de ses utilitaires proviennent des distributions BSD. Les distributions Linux sont également très dépendantes du compilateur, des débogueurs et d'autres outils de développements écrits par la fondation pour les logiciels libres, la FSF. Ensemble, le CSRG, la FSF, et les développeurs du noyau du système Linux ont créé la plate-forme à partir de laquelle le mouvement des logiciels libres Open Source a été lancé. Je suis fier d'avoir eu l'occasion d'aider les pionniers de ce mouvement. J'attends avec impatience le jour où tous les utilisateurs et toutes les sociétés préféreront développer et acheter des logiciels de ce type.

Chapitre 4

L'Internet Engineering Task Force

Pour une entité purement virtuelle, l'Internet Engineering Task Force (IETF, le groupe de travail de génie Internet) a eu un impact d'une importance capitale. Hormis le protocole TCP/IP lui-même l'IETF développa ou améliora toutes les techniques de l'Internet. Les groupes de travail de l'IETF ont créé les standards de routage, de gestion de réseaux et du transport sans lesquels l'Internet n'existerait pas. Ils ont défini les standards de sécurité qui rendront l'Internet plus sûr, les standards de qualité de service qui feront de l'Internet un environnement plus prévisible et le standard de la prochaine génération du protocole Internet lui-même.

Ces standards ont connu une réussite prodigieuse. L'Internet se développe plus vite que n'importe quel autre outil connu, bien plus vite que ne l'ont fait le chemin de fer, l'électricité, le téléphone ou la télévision et ce n'est encore qu'un début. Tout cela s'est fait grâce à des actions volontaires : aucun gouvernement n'exige l'utilisation de standards IETF. Des standards concurrents, certains émanant de divers gouvernements, ont fait de brèves apparitions, alors que les standards IETF prospèrent. Mais tous les standards IETF n'ont pas réussi. Seuls les standards qui répondent à des exigences issues du monde réel et ce, correctement, deviennent les vrais standards, pas seulement sur le papier.

La réussite de l'IETF et le décollage de la communauté du logiciel libre relèvent des mêmes causes. Les standards IETF sont développés selon un processus ouvert et transparent, auquel toute personne intéressée peut prendre part. Tous les documents de l'IETF sont librement accessibles sur l'Internet et peuvent être reproduits à volonté. En fait, le processus de documentation ouvert de l'IETF constitue un cas d'école du potentiel du mouvement du logiciel libre.

Cet essai propose tout d'abord un bref historique de l'IETF. Nous décrirons ensuite son organisation et ses processus de création de standards. Nous proposerons pour finir quelques réflexions supplémentaires sur l'importance des standards ouverts, des documents ouverts et du logiciel libre.

4.1 L'histoire de l'IETF

L'IETF est apparue en janvier 1986, sous la forme d'une rencontre trimestrielle de chercheurs subventionnés par le gouvernement américain. À partir de la quatrième rencontre de l'IETF, en octobre de la même année, des représentants d'entreprises privées furent invités. Depuis lors, toutes les rencontres de l'IETF sont ouvertes à quiconque veut y participer. Les premières rencontres se faisaient en petit comité, moins de 35 participants à chacune des cinq premières rencontres, avec un maximum sur les treize premières rencontres de seulement 120 participants (à la douzième, en janvier 1989). L'IETF s'est bien développée depuis lors, avec plus 500 participants à la 23$^{\text{ème}}$ rencontre (mars 1994), plus de 1 000 à la 31$^{\text{ème}}$ (décembre 1994) et presque 2 000 à la 37$^{\text{ème}}$ (décembre 1996). Le taux de croissance du nombre des participants a ensuite ralenti, 2 100 personnes assistèrent à la 43$^{\text{ème}}$ rencontre (décembre 1998). En cours de route, en 1991, le nombre de rencontres est passé de quatre à trois par an.

L'IETF dispose d'un petit secrétariat, actuellement installé à Reston (Virginie), et d'un service d'édition des RFC, actuellement pris en charge par l'Institut des Sciences de l'Information de l'université de Californie du Sud.

L'IETF elle-même n'a jamais eu le statut d'une entité légale. C'est tout simplement une activité sans fondement légal. Jusqu'à fin 1997 l'IETF fonctionnait

grâce à des fonds gouvernementaux et des droits d'entrée aux rencontres. Depuis début 1998, l'Internet Society a pris le relais du gouvernement.

L'Internet Society, a été constituée en 1992 en partie pour fournir un parapluie légal au processus de standardisation de l'IETF et aussi dans le but de réunir des fonds pour ses activités. Cette organisation associative internationale à but non lucratif répand partout la Bonne Parole de l'Internet, même dans des parties du monde encore non connectées. À l'heure actuelle, l'IETF peut être au mieux décrite comme une instance de développement de standards opérant sous les auspices de l'Internet Society.

L'idée de groupes de travail a été introduite lors de la cinquième rencontre de l'IETF (février 1987), et à présent plus de 110 de ces groupes fonctionnent.

4.2 Structures et caractéristiques de l'IETF

L'IETF peut être au mieux décrite comme une organisation associative à laquelle l'appartenance ne serait pas définie. Le seul critère de sélection est la restriction aux personnes physiques (à l'exclusion des organisations et des sociétés) de la possibilité d'être membres de l'IETF. Toute personne qui fait partie d'une liste de diffusion de l'IETF ou qui assiste à une rencontre de l'IETF peut être considérée comme un membre de l'IETF.

L'IETF compte maintenant 115 groupes de travail officiels organisés en huit sections : applications, générale, Internet, opérations et gestion, routage, sécurité, transport, services aux utilisateurs. Chacune de ces sections est dirigée par un ou deux directeurs de section, volontaires. Les directeurs de section réunis en groupe, avec le président de l'IETF, constituent l'Internet Engineering Steering Group (IESG, Groupe de pilotage de l'IETF). L'IESG est le bureau d'approbation des standards pour l'IETF. Il y a en outre un Internet Architecture Board, de 12 membres, qui conseille l'IESG pour la formation des groupes de travail et sur l'organisation de leurs travaux.

Les membres de l'IAB et les directeurs de section sont choisis — leur mandat dure deux ans — par un comité de nomination composé chaque année de membres tirés au sort parmi des volontaires ayant participé à au moins deux des trois précédentes rencontres.

4.3 Les groupes de travail de l'IETF

L'une des différences essentielles entre l'IETF et beaucoup d'autres organismes de standardisation est que l'IETF est surtout dirigée par sa base. Il est tout-à-fait rare que l'IESG ou l'IAB créent un groupe de travail de leur propre chef pour travailler à un problème qui paraît important. Presque tous les groupes de travail sont constitués quand un petit groupe de personnes intéressées se rassemble, de leur propre initiative et propose de former un groupe de travail à un directeur de section. Cela signifie que l'IETF ne peut pas établir de plans de travail pour l'avenir, mais cela contribue aussi à la certitude qu'un enthousiasme et des compétences suffisantes assureront sa réussite.

Le directeur de section travaille avec les personnes qui veulent créer un groupe de travail à développer une certaine charte. La charte de chaque groupe établit la liste des documents qu'il devra produire, des activités de liaison qu'il

pourra être amené à entretenir avec d'autres ainsi que des limites du domaine qu'il explorera. Ce dernier point s'avère souvent le plus important. La proposition de charte circule alors dans les listes de diffusion de l'IESG et de l'IAB, pour commentaires. Si aucun problème significatif n'est soulevé dans la semaine, la charte est postée dans la liste publique de l'IETF et dans une liste de liaison avec les autres organismes de standardisation, pour savoir si des travaux dont l'IETF devrait avoir connaissance sont en cours dans d'autres forums. Après une autre semaine, destinée à recueillir d'éventuelles remarques complémentaires, l'IESG peut alors approuver la charte et créer le groupe de travail.

4.4 Les documents de l'IETF

Tous les documents de de l'IETF sont publics et librement accessibles sur l'Internet. Les auteurs concèdent à l'IETF un copyright limité lorsque les documents sont publiés, ce copyright assurant que les documents restent librement accessibles (l'auteur ne pourra pas décider de retirer un jour un document), sont republiables dans leur intégralité par quelqu'un d'autre et pour la plupart d'entre eux, que l'IETF peut les exploiter dans l'élaboration de standards à venir. L'auteur conserve tous les autres droits.

La série éditoriale de base de l'IETF est la série des RFC. RFC signifie en principe Request for Comments mais RFC, aujourd'hui, ne signifie rien d'autre que "RFC" car les documents publiés dans la série sont généralement passés par un long processus de révision avant leur publication. Les publications RFC se divisent en deux grandes catégories : standard et non standard. Les publications standard peuvent avoir les statuts de : proposition de standard, pré-standard, standard de l'Internet. Les publications non standard se classent de la manière suivante : informatives, expérimentales, historiques.

Outre les RFC, l'IETF a recours aux « brouillons Internet » [1]. Ce sont des documents temporaires dont la finalité est proche de celle des premières RFC (request for comment) et qui sont automatiquement retirés après six mois. Les « brouillons Internet » ne doivent pas être cités ou faire l'objet d'une référence quelle qu'elle soit sauf dans le cadre de l'enseignement.

4.5 Le processus IETF

La devise de l'IETF est : « Le consensus sur l'essentiel, et du code qui fonctionne ». L'unanimité du groupe de travail n'est pas indispensable à l'adoption d'une proposition mais une proposition qui ne rallie pas la majorité des membres du groupe ne sera pas acceptée. Il n'est pas requis qu'une proposition soit défendue par un pourcentage donné des membres du groupe mais la plupart des propositions qui sont défendues par plus de 90 % des membres ont des chances d'être acceptées, quant à celles qui sont défendues par moins de 80 % des membres, elles sont souvent rejetées. Les groupes de travail de l'IETF ne votent pas vraiment mais peuvent recourir à un vote à main levée pour vérifier que le consensus est atteint.

Les documents de type non standard peuvent émaner de l'activité des groupes de travail de l'IETF, ou de personnes indépendantes souhaitant que leurs

1. Internet drafts

réflexions ou leurs propositions techniques soient accessibles à la communauté de l'Internet. Presque toutes les propositions pour une publication dans les RFC sont revues par l'IESG, qui conseillera ensuite l'éditeur des RFC quant à l'intérêt de la publication du document. L'éditeur des RFC décidera alors de publier ou non le document et si l'IESG a rédigé une note, de l'inclure ou non dans le document. Ces notes de l'IESG servent à marquer certaines réserves par rapport à la proposition, l'IESG ayant alors le sentiment qu'un avertissement quelconque à l'utilisateur peut être utile.

Pour un document standard, normalement, un groupe de travail de l'IETF produira un « brouillon Internet », destiné à être publié dans les RFC. L'étape finale de l'évaluation de la proposition au sein du groupe de travail est un « dernier appel », qui dure en principe deux semaines, pendant lequel le président du groupe de travail demande dans la liste de diffusion du groupe si la proposition connaît des problèmes notables. Si, à la suite de ce « dernier appel », il apparaît que le groupe est d'accord sur l'acceptation de la proposition, celle-ci est alors transmise à l'IESG, pour évaluation. La première étape de l'évaluation par l'IESG consiste en un « dernier appel » à l'échelle de l'IETF, envoyé à la principale liste de diffusion de l'IETF. Les personnes qui n'ont pas suivi les travaux du groupe peuvent ainsi apporter leurs commentaires à la proposition. En principe, ce « dernier appel » dure deux semaines pour les propositions qui viennent de groupes de travail de l'IETF et quatre semaines pour les autres.

L'IESG prend les résultats du « dernier appel » à l'IETF comme base pour ses délibérations au sujet de la proposition. L'IESG peut approuver le document et demander sa publication ou renvoyer la proposition à son (ses) auteur(s) pour des révisions fondées sur l'évaluation qu'elle en a faite. C'est ce même processus qui est utilisé à chaque étape pour les documents standard.

Normalement, les propositions sont intégrées aux documents standard au titre de standards proposés mais lorsque des incertitudes pèsent quant à l'adéquation de l'approche ou si une phase de tests supplémentaire paraît nécessaire, un document est d'abord publié sous statut expérimental. Les standards proposés sont censés être de bonnes idées sans problème technique connu. Après six mois au minimum comme standard proposé, l'accession d'une proposition au statut de pré-standard peut être discutée. Les pré-standards doivent avoir fait preuve de la clarté de leur documentation et démontré que tous les problèmes de droit d'auteur éventuels sont repérés, et solubles. On y arrive en exigeant au moins deux implémentations génétiquement indépendantes et inter-opérables de la proposition, avec, le cas échéant, des exercices séparés de procédures d'obtention de licences. Ce qui implique aussi que toutes les caractéristiques séparées du protocole soient implémentées à plusieurs reprises. Toute caractéristique ne répondant pas à ces exigences doit être retirée avant que le processus puisse continuer à se dérouler. Les standards IETF peuvent donc se simplifier au fur et à mesure qu'ils progressent. On retrouve la seconde moitié de la devise de l'IETF : « du code qui fonctionne ».

La dernière étape du processus de standardisation IETF est le standard Internet. Si la proposition a connu un succès commercial significatif après au moins quatre mois au statut de pré-standard, son accession au statut de standard Internet peut être discutée.

Deux différences essentielles sont notables si l'on compare le type standard de l'IETF avec la manière dont les choses se passent dans d'autres organisations de standardisation. D'abord, le résultat final de la plupart des processus

de standardisation est à peu près équivalent au statut de standard proposé pour l'IETF. Une bonne idée mais pas d'exigence d'un code qui fonctionne vraiment. Ensuite, le consensus sur l'essentiel au lieu de l'unanimité peut donner des propositions avec moins de caractéristiques ajoutées destinées à faire taire une objection particulière.

En bref, l'IETF fonctionne selon un schéma de travail dirigé par la base et croit aux vertus du principe « essayez avant d'acheter ».

4.6 Standards ouverts, documents ouverts, et logiciel libre

La documentation ouverte et la politique de développement des standards de l'IETF restent les raisons principales du succès des standards IETF. L'IETF est l'une des rares grandes organisations de standardisation produisant des documents librement accessibles, comme toutes ses listes de diffusion et ses rencontres. Dans la plupart des organisations de standardisation traditionnelles et même dans quelques-uns des groupes, plus récents, liés à l'Internet, l'accès aux documents et aux rencontres est réservé aux membres, ou n'est possible qu'après paiement d'une certaine somme. Parfois parce que l'organisation emploie une partie des fonds recueillis pour elle-même, à travers la vente de ses standards. Dans d'autres cas, il s'agit d'organisations dont les membres paient une cotisation. Il faut bien entendu être en mesure de participer au processus de développement des standards et d'y avoir accès au cours de leur développement.

Les standards mis au point par un comité restreint ne répondent souvent pas aussi bien qu'ils le devraient aux besoins de l'utilisateur ou des entreprises, ou sont plus complexes que ce que la communauté des opérateurs peut raisonnablement tolérer. Restreindre l'accès aux documents en cours d'élaboration rend plus difficile pour celui qui implémentent les standards de comprendre la genèse et la raison de telle caractéristique spécifique et cela peut conduire à des aberrations. Restreindre l'accès aux standards terminaux rend impossible aux étudiants et aux développeurs des petites start-up de les comprendre, et donc de les utiliser.

L'IETF a soutenu le concept de logiciel libre bien avant que le mouvement du logiciel libre se soit formé. Jusqu'à récemment, en principe, les « implémentations de référence » de standards IETF faisaient partie intégrante de l'exigence d'implémentations multiples liée à l'avancement du processus de standardisation. Cela n'a jamais été formellement une partie du processus de l'IETF, mais fut généralement un très utile effet secondaire. Le mouvement s'est malheureusement un peu ralenti à notre époque de standards plus complexes, et de plus grandes implications économiques pour les standards. La pratique ne s'en est jamais arrêtée, mais ce serait une bonne chose si le mouvement du logiciel libre pouvait revigorer cette part non officielle du processus de standardisation de l'IETF.

Il se peut que cela ne soit pas immédiatement apparent, mais l'accessibilité des processus et de la documentation des standards ouverts est vitale pour le mouvement du logiciel libre. Sans un clair accord au sujet de ce sur quoi l'on travaille, normalement défini dans la documentation des standards, il est assez facile pour des projets de développement distribué, comme c'est le cas du

mouvement du logiciel libre, de se fragmenter et de faire fausse route. Un partenariat intrinsèque s'établit entre les participants aux processus d'élaboration de ces standards, la documentation ouverte, et le logiciel libre. Ce partenariat a créé l'Internet, et créera d'autres merveilles encore à l'avenir.

Chapitre 5

Le système d'exploitation du projet GNU et le mouvement du logiciel libre

5.1 La première communauté qui partageait le logiciel

En 1971, quand j'ai commencé à travailler au laboratoire d'intelligence artificielle (IA) du MIT [1], j'ai intégré une communauté qui partageait le logiciel depuis de nombreuses années déjà. Le partage du logiciel n'était pas limité à notre communauté ; c'est une notion aussi ancienne que les premiers ordinateurs, tout comme on partage des recettes depuis les débuts de la cuisine. Mais nous partagions davantage que la plupart.

Le laboratoire d'IA utilisait un système d'exploitation à temps partagé appelé ITS (Incompatible Timesharing System, ou système à temps partagé incompatible) que les hackers [2] de l'équipe avaient écrit et mis au point en langage d'assemblage pour le Digital PDP-10, l'un des grands ordinateurs de l'époque. En tant que membre de cette communauté, hacker système de l'équipe du laboratoire d'IA, mon travail consistait à améliorer ce système.

Nous ne qualifiions pas nos productions de « logiciels libres », car ce terme n'existait pas encore ; c'est pourtant ce qu'elles étaient. Quand d'autres universitaires ou quand des ingénieurs souhaitaient utiliser et porter l'un de nos programmes, nous les laissions volontiers faire. Et quand on remarquait que quelqu'un utilisait un programme intéressant mais inconnu, on pouvait toujours en obtenir le code source, afin de le lire, le modifier, ou d'en réutiliser des parties dans le cadre d'un nouveau programme.

5.2 L'effondrement de la communauté

La situation s'est modifiée de manière drastique au début des années 1980 quand la société Digital™ a mis fin à la série des PDP-10. Cette architecture, élégante et puissante dans les années 60, ne pouvait pas s'étendre naturellement aux plus grands espaces d'adressage qu'on trouvait dans les années 80. Cela rendait obsolètes la quasi-totalité des programmes constituant ITS.

La communauté des hackers du laboratoire d'IA s'était effondrée peu de temps auparavant. La plupart d'entre eux avaient été engagés par une nouvelle société, Symbolics™, et ceux qui étaient restés ne parvenaient pas à maintenir la communauté (le livre Hackers, écrit par Steve Levy, décrit ces événements et dépeint clairement l'apogée de cette communauté). Quand le laboratoire a, en 1982, choisi d'acheter un nouveau PDP-10, ses administrateurs ont décidé de remplacer ITS par le système de temps partagé de la société Digital™, qui n'était pas libre.

Les ordinateurs modernes d'alors, tels que le VAX ou le 68020, disposaient de leurs propres systèmes d'exploitation, mais aucun d'entre eux n'était un logiciel libre : il fallait signer un accord de non divulgation pour en obtenir ne serait-ce que des copies exécutables.

1. Institut de Technologie du Massachusetts, l'une des universités les plus prestigieuses des États-Unis d'Amérique

2. L'utilisation du mot « hacker » dans le sens de « qui viole des systèmes de sécurité » est un amalgame instillé par les mass media. Nous autres hackers refusons de reconnaître ce sens, et continuons d'utiliser ce mot dans le sens « qui aime programmer et apprécie de le faire de manière astucieuse et intelligente. » (En français, on peut utiliser le néologisme « bitouilleur » pour désigner l'état d'esprit de celui qui « touille des bits »)

Cela signifiait que la première étape de l'utilisation d'un ordinateur était de promettre de ne pas aider son prochain. On interdisait toute communauté coopérative. La règle qu'édictaient ceux qui détenaient le monopole d'un logiciel propriétaire était « Qui partage avec son voisin est un pirate. Qui souhaite la moindre modification doit nous supplier de la lui faire ».

L'idée que le système social du logiciel propriétaire — le système qui vous interdit de partager ou d'échanger le logiciel — est antisocial, immoral, et qu'il est tout bonnement incorrect, surprendra peut-être certains lecteurs. Mais comment qualifier autrement un système fondé sur la division et l'isolement des utilisateurs ? Les lecteurs surpris par cette idée ont probablement pris le système social du logiciel propriétaire pour argent comptant, ou l'ont jugé en employant les termes suggérés par les entreprises vivant de logiciels propriétaires. Les éditeurs de logiciels travaillent durement, et depuis longtemps, pour convaincre tout un chacun qu'il n'existe qu'un seul point de vue sur la question — le leur.

Quand les éditeurs de logiciels parlent de « faire respecter » leurs « droits » ou de « couper court au piratage », le contenu réel de leur discours passe au second plan. Le véritable message se trouve entre les lignes, et il consiste en des hypothèses de travail qu'ils considèrent comme acquises ; nous sommes censés les accepter les yeux fermés. Passons-les donc en revue.

La première hypothèse est que les sociétés éditrices de logiciels disposent d'un droit naturel, incontestable, à être propriétaire du logiciel et asseoir ainsi leur pouvoir sur tous ses utilisateurs (si c'était là un droit naturel, on ne pourrait formuler aucune objection, indépendamment du tort qu'il cause à tous). Il est intéressant de remarquer que la constitution et la tradition juridique des États-Unis d'Amérique rejettent toutes deux cette idée ; le copyright n'est pas un droit naturel, mais un monopole artificiel, imposé par l'État, qui restreint le droit naturel qu'ont les utilisateurs de copier le logiciel.

Une autre hypothèse sous-jacente est que seules importent les fonctionnalités du logiciel — et que les utilisateurs d'ordinateurs n'ont pas leur mot à dire quant au modèle de société qu'ils souhaitent voir mettre en place.

Une troisième hypothèse est qu'on ne disposerait d'aucun logiciel utilisable (ou qu'on ne disposerait jamais d'un logiciel qui s'acquitte de telle ou telle tâche en particulier) si on ne garantissait pas à une société l'assise d'un pouvoir sur les utilisateurs du programme. Cette idée était plausible, jusqu'à ce que le mouvement du logiciel libre démontrât qu'on peut produire toutes sortes de logiciels utiles sans qu'il soit nécessaire de les barder de chaînes.

Si l'on se refuse à accepter ces hypothèses, et si on examine ces questions en se fondant sur une moralité dictée par le bon sens, qui place les utilisateurs au premier plan, on parvient à des conclusions bien différentes. Les utilisateurs des ordinateurs doivent être libres de modifier les programmes pour qu'ils répondent mieux à leurs besoins, et libres de partager le logiciel, car la société est fondée sur l'aide à autrui.

La place me manque ici pour développer le raisonnement menant à cette conclusion, aussi renverrai-je le lecteur au document web *Why Software Should Not Have Owners* (pourquoi le logiciel ne devrait appartenir à personne).

5.3 Une profonde prise de décision

Avec la fin de ma communauté, il m'était impossible de continuer comme de par le passé. J'étais au lieu de cela confronté à une profonde prise de décision.

La solution de facilité était de rejoindre le monde du logiciel propriétaire, de signer des accords de non divulgation et promettre ainsi de ne pas aider mon ami hacker. J'aurais aussi été, très probablement, amené à développer du logiciel qui aurait été publié en fonction d'exigences de non divulgation, augmentant la pression qui en inciterait d'autres à trahir également leurs semblables.

J'aurais pu gagner ma vie de cette manière, et peut-être me serais-je même amusé à écrire du code. Mais je savais qu'à la fin de ma carrière, je n'aurais à contempler que des années de construction de murs pour séparer les gens, et que j'aurais l'impression d'avoir employé ma vie à rendre le monde pire.

J'avais déjà eu l'expérience douloureuse des accords de non divulgation, quand quelqu'un m'avait refusé, ainsi qu'au laboratoire d'IA du MIT, l'accès au code source du programme de contrôle de notre imprimante (l'absence de certaines fonctionnalités dans ce programme rendait l'utilisation de l'imprimante très frustrante). Aussi ne pouvais-je pas me dire que les accords de non divulgation étaient bénins. J'avais été très fâché du refus de cette personne de partager avec nous ; je ne pouvais pas, moi aussi, me comporter d'une telle manière à l'égard de mon prochain.

Une autre possibilité, radicale mais déplaisante, était d'abandonner l'informatique. De cette manière, mes capacités ne seraient pas employées à mauvais escient, mais elles n'en seraient pas moins gaspillées. Je ne me rendrais pas coupable de diviser et de restreindre les droits des utilisateurs d'ordinateurs, mais cela se produirait malgré tout.

Alors, j'ai cherché une façon pour un programmeur de se rendre utile pour la bonne cause. Je me suis demandé si je ne pouvais pas écrire un ou plusieurs programmes qui permettraient de souder à nouveau une communauté.

La réponse était limpide : le besoin le plus pressant était un système d'exploitation. C'est le logiciel le plus crucial pour commencer à utiliser un ordinateur. Un système d'exploitation ouvre de nombreuses portes ; sans système, l'ordinateur est inexploitable. Un système d'exploitation libre rendrait à nouveau possible une communauté de hackers, travaillant en mode coopératif — et tout un chacun serait invité à participer. Et tout un chacun pourrait utiliser un ordinateur sans devoir adhérer à une conspiration visant à priver ses amis des logiciels qu'il utilise.

En tant que développeur de système d'exploitation, j'avais les compétences requises. Aussi, bien que le succès ne me semblât pas garanti, j'ai pensé être le candidat de choix pour ce travail. J'ai décidé de rendre le système compatible avec Unix™ de manière à le rendre portable, et pour le rendre plus accessible aux utilisateurs d'Unix™. J'ai opté pour le nom GNU, fidèle en cela à une tradition des hackers, car c'est un acronyme récursif qui signifie « GNU's Not Unix » (GNU N'est pas Unix).

Un système d'exploitation ne se limite pas à un noyau, qui suffit à peine à exécuter d'autres programmes. Dans les années 1970, tout système d'exploitation digne de ce nom disposait d'interpréteurs de commandes, d'assembleurs, de compilateurs, d'interpréteurs, de débogueurs, d'éditeurs de textes, de logiciels de courrier électronique, pour ne citer que quelques exemples. C'était le cas d'ITS,

c'était le cas de Multics™, c'était le cas de VMS™, et c'était le cas d'Unix™. Ce serait aussi le cas du système d'exploitation GNU.

Plus tard, j'ai entendu ces mots, attribués à Hillel[3] [4] :

> If I am not for myself, who will be for me ?
> If I am only for myself, what am I ?
> If not now, when ?

C'est dans cet état d'esprit que j'ai pris la décision de lancer le projet GNU.

5.4 Free comme liberté

Parfois[5], le terme « free software » est mal compris — il n'a rien à voir avec le prix. Il parle de liberté. Voici donc la définition d'un logiciel libre : un programme est un logiciel libre pour vous, utilisateur en particulier, si :

- Vous avez la liberté de l'exécuter, pour quelque motif que ce soit,
- Vous avez la liberté de modifier le programme afin qu'il corresponde mieux à vos besoins (dans la pratique, pour que cette liberté prenne effet, il vous faut pouvoir accéder au code source, puisqu'opérer des modifications au sein d'un programme dont on ne dispose pas du code source est un exercice extrêmement difficile),
- Vous disposez de la liberté d'en redistribuer des copies, que ce soit gratuitement ou contre une somme d'argent,
- Vous avez la liberté de distribuer des versions modifiées du programme, afin que la communauté puisse bénéficier de vos améliorations.

Puisque le mot « free » se réfère ici à la liberté, et non au prix, il n'est pas contradictoire de vendre des copies de logiciels libres. En réalité, cette liberté est cruciale : les compilations de logiciels libres vendues sur CD-ROM sont importantes pour la communauté, et le produit de leur vente permet de lever des fonds pour le développement du logiciel libre. C'est pourquoi on ne peut pas qualifier de logiciel libre un logiciel qu'on n'a pas la liberté d'inclure dans de telles compilations.

Le mot « free » étant ambigu, on a longtemps cherché des solutions de remplacement, mais personne n'en a trouvé de convenable. La langue anglaise compte plus de mots et de nuances que toute autre langue, mais elle souffre de l'absence d'un mot simple, univoque, qui ait le sens de « free », comme liberté — « unfettered » (terme littéraire signifiant « sans entrave ») étant le meilleur candidat, d'un point de vue sémantique, des mots comme « liberated » (« libéré »), « freedom » (« liberté »), et « open » (« ouvert ») présentant tous un sens incorrect ou un autre inconvénient.

3. En tant qu'athée, je ne suis les pas d'aucun meneur religieux, mais j'admire parfois ce que l'un d'entre eux a dit.

4. on peut rendre l'esprit de ce poème comme suit :

> Si je ne suis rien pour moi-même, qui sera pour moi ?
> Si je suis tout pour moi-même, que suis-je ?
> Si ce n'est pas aujourd'hui, alors quand ?

5. en anglais, le « libre » de « logiciel libre » se dit « free ». Malheureusement, ce mot a une autre acception, indépendante et incorrecte ici, il signifie également « gratuit ». Cette ambiguïté a causé énormément de tort au mouvement du logiciel libre.

5.5 Les logiciels et le système du projet GNU

C'est une gageure que de développer un système complet. Pour mener ce projet à bien, j'ai décidé d'adapter et de réutiliser les logiciels libres existants, quand cela était possible. J'ai par exemple décidé dès le début d'utiliser TEX comme formateur de texte principal ; quelques années plus tard, j'ai décidé d'utiliser le système de fenêtrage X plutôt que d'écrire un autre système de fenêtrage pour le projet GNU.

Cette décision a rendu le système GNU distinct de la réunion de tous les logiciels GNU. Le système GNU comprend des programmes qui ne sont pas des logiciels GNU, ce sont des programmes qui ont été développés par d'autres, dans le cadre d'autres projets, pour leurs buts propres, mais qu'on peut réutiliser, car ce sont des logiciels libres.

5.6 La genèse du projet

En janvier 1984, j'ai démissionné de mon poste au MIT et j'ai commencé à écrire les logiciels du projet GNU. Il était nécessaire que je quitte le MIT pour empêcher ce dernier de s'immiscer dans la distribution du projet GNU en tant que logiciel libre. Si j'étais resté dans l'équipe, le MIT aurait pu se déclarer le propriétaire de mon travail, et lui imposer ses propres conditions de distribution, voire en faire un paquetage de logiciels propriétaires. Je n'avais pas l'intention d'abattre autant de travail et de le voir rendu inutilisable pour ce à quoi il était destiné : créer une nouvelle communauté qui partage le logiciel.

Cependant, le professeur Winston, qui dirigeait alors le laboratoire d'IA du MIT, m'a gentiment invité à continuer à utiliser les équipements du laboratoire.

5.7 Les premiers pas

Peu de temps avant de me lancer dans le projet GNU, j'avais entendu parler du Free University Compiler Kit [6], plus connu sous le nom de VUCK (en hollandais, le mot « free » commence par un V). Ce compilateur avait été mis au point dans l'intention de traiter plusieurs langages, parmi lesquels C et Pascal, et de produire des binaires pour de nombreuses machines cibles. J'ai écrit à son auteur en lui demandant la permission d'utiliser ce compilateur dans le cadre du projet GNU.

Il répondit d'un ton railleur, en déclarant (en anglais) que l'université était « free » [7], mais pas le compilateur. J'ai alors décidé que le premier programme du projet GNU serait un compilateur traitant de plusieurs langages, sur plusieurs plates-formes.

En espérant m'épargner la peine d'écrire tout le compilateur moi-même, j'ai obtenu le code source du compilateur Pastel, qui était un compilateur pour plusieurs plates-formes, développé au laboratoire Lawrence Livermore. Il compilait, et c'était aussi le langage dans lequel il avait été écrit, une version étendue

6. En anglais, le placement des mots ne permet pas de déterminer s'il s'agit d'un « kit compilateur libre de l'université » ou d'un « kit compilateur de l'université libre ».

7. M. Stallman ne sait pas s'il voulait ici dire « libre » ou « gratuite ».

de Pascal, mise au point pour jouer le rôle de langage de programmation système. J'y ai ajouté une interface pour le C, et j'ai entrepris le portage de ce programme sur le Motorola 68000. Mais j'ai dû abandonner quand j'ai découvert que ce compilateur ne fonctionnait qu'avec plusieurs méga-octets d'espace de pile disponibles, alors que le système Unix du 68000 ne gérait que 64 Ko d'espace de pile.

C'est alors que j'ai compris que le compilateur Pastel avait été mis au point de telle manière qu'il analysait le fichier en entrée, en faisait un arbre syntaxique, convertissait cet arbre syntaxique en chaîne d'« instructions », et engendrait ensuite le fichier de sortie, sans jamais libérer le moindre espace mémoire occupé. J'ai alors compris qu'il me faudrait réécrire un nouveau compilateur en partant de zéro. Ce compilateur est maintenant disponible, il s'appelle GCC ; il n'utilise rien du compilateur Pastel, mais j'ai réussi à adapter et à réutiliser l'analyseur syntaxique que j'avais écrit pour le C. Mais tout cela ne s'est produit que quelques années plus tard ; j'ai d'abord travaillé sur GNU Emacs.

5.8 GNU Emacs

J'ai commencé à travailler sur GNU Emacs en septembre 1984, et ce programme commençait à devenir utilisable début 1985. Cela m'a permis d'utiliser des systèmes Unix pour éditer mes fichiers ; vi et ed me laissant froid, j'avais jusqu'alors utilisé d'autres types de machines pour éditer mes fichiers.

C'est alors que j'ai reçu des requêtes de gens souhaitant utiliser GNU Emacs, ce qui a soulevé le problème de sa distribution. Je l'avais bien sûr proposé sur le serveur ftp de l'ordinateur du MIT que j'utilisais (cet ordinateur, prep.ai.mit.edu, a ainsi été promu au rang de site de distribution par ftp principal du projet GNU ; quelques années plus tard, à la fin de son exploitation, nous avons transféré ce nom sur notre nouveau serveur ftp). Mais à l'époque, une proportion importante des personnes intéressées n'avaient pas d'accès à l'Internet et ne pouvaient pas obtenir une copie du programme par ftp. La question se posait en ces termes : que devais-je leur dire ?

J'aurais pu leur dire : « Trouvez un ami qui dispose d'un accès au réseau et qui vous fera une copie. » J'aurais pu également procéder comme j'avais procédé avec la version originale d'Emacs, sur PDP-10, et leur dire : « Envoyez-moi une bande et une enveloppe timbrée auto-adressée, et je vous les renverrai avec Emacs. » Mais j'étais sans emploi, et je cherchais des moyens de gagner de l'argent grâce au logiciel libre. C'est pourquoi j'ai annoncé que j'enverrais une bande à quiconque en désirait une, en échange d'une contribution de 150 USD. De cette manière, je mettais en place une entreprise autour du marché de la distribution du logiciel libre, entreprise précurseur des sociétés qu'on trouve aujourd'hui et qui distribuent des systèmes GNU entiers fondés sur Linux.

5.9 Un programme est-il libre pour chacun de ses utilisateurs ?

Si un programme est un logiciel libre au moment où il quitte les mains de son auteur, cela ne signifie pas nécessairement qu'il sera un logiciel libre pour quiconque en possédera une copie. Le logiciel relevant du domaine public,

par exemple (qui ne tombe sous le coup d'aucun copyright), est du logiciel libre ; mais tout un chacun peut en produire une version propriétaire modifiée. De façon comparable, de nombreux programmes libres sont couverts par des copyrights mais distribués sous des licences permissives qui autorisent la création de versions modifiées et propriétaires.

L'exemple le plus frappant de ce problème est le système de fenêtrage X. Développé au MIT, et distribué sous forme de logiciel libre sous une licence permissive, il a rapidement été adopté par divers constructeurs. Ils ont ajouté X à leurs systèmes Unix™ propriétaires, sous forme binaire uniquement, en le frappant du même accord de non divulgation. Ces exemplaires de X n'étaient en rien du logiciel plus libre que le reste d'Unix™.

Les développeurs du système de fenêtrage X ne voyaient là nul problème — ils s'attendaient à cela et souhaitaient un tel résultat. Leur but n'était pas la liberté, mais la simple « réussite », définie comme le fait d'« avoir beaucoup d'utilisateurs. » Peu leur importait la liberté de leurs utilisateurs, seul leur nombre revêtait de l'importance à leurs yeux.

Cela a conduit à une situation paradoxale, où deux différentes façons d'évaluer la liberté répondaient de manières différente à la question « Ce programme est-il libre ? » Qui fondait son jugement sur la liberté fournie par les conditions de distribution de la distribution du MIT, concluait que X était un logiciel libre. Mais qui mesurait la liberté de l'utilisateur type de X, devait conclure que X était un logiciel propriétaire. La plupart des utilisateurs de X exécutaient des versions propriétaires fournies avec des systèmes Unix™, et non la version libre.

5.10 Le copyleft et la GPL de GNU

Le but du projet GNU était de rendre les utilisateurs libres, pas de se contenter d'être populaire. Nous avions besoins de conditions de distribution qui empêcheraient de transformer du logiciel GNU en logiciel propriétaire. Nous avons utilisé la méthode du copyleft [8], ou « gauche d'auteur ».

Le gauche d'auteur utilise les lois du droit d'auteur, en les retournant pour leur faire servir le but opposé de ce pour quoi elles ont été conçues : ce n'est pas une manière de privatiser du logiciel, mais une manière de le laisser « libre ».

L'idée centrale du gauche d'auteur est de donner à quiconque la permission d'exécuter le programme, de le copier, de le modifier, et d'en distribuer des versions modifiées — mais pas la permission d'ajouter des restrictions de son cru. C'est ainsi que les libertés cruciales qui définissent le « logiciel libre » sont garanties pour quiconque en possède une copie ; elles deviennent des droits inaliénables.

Pour que le gauche d'auteur soit efficace, il faut que les versions modifiées demeurent libres, afin de s'assurer que toute œuvre dérivée de notre travail reste disponible à la communauté en cas de publication. Quand des programmeurs professionnels se portent volontaires pour améliorer le logiciel GNU, c'est le gauche d'auteur qui empêche leurs employeurs de dire : « Vous ne pouvez pas

8. En 1984 ou 1985, Don Hopkins (dont l'imagination était sans bornes) m'a envoyé une lettre. Il avait écrit sur l'enveloppe plusieurs phrases amusantes, et notamment celle-ci : « Copyleft — all rights reversed. » (« couvert par le gauche d'auteur, tous droits renversés. »). J'ai utilisé le mot « copyleft » pour donner un nom au concept de distribution que je développais alors.

partager ces modifications, car nous allons les utiliser dans le cadre de notre version propriétaire du programme. »

Il est essentiel d'imposer que les modifications restent libres si on souhaite garantir la liberté de tout utilisateur du programme. Les sociétés qui ont privatisé le système de fenêtrage X faisaient en général quelques modifications pour le porter sur leurs systèmes et sur leur matériel. Ces modifications étaient ténues si on les comparait à X dans son ensemble, mais elles n'en étaient pas pour autant faciles. Si le fait de procéder à des modifications pouvait servir de prétexte à ôter leur liberté aux utilisateurs, il serait facile pour quiconque de s'en servir à son avantage.

Le problème de la réunion d'un programme libre avec du code non libre est similaire. Une telle combinaison serait indubitablement non libre ; les libertés absentes de la partie non libre du programme ne se trouveraient pas non plus dans l'ensemble résultat de cette compilation. Autoriser de telles pratiques ouvrirait une voie d'eau suffisante pour couler le navire. C'est pourquoi il est crucial pour le gauche d'auteur d'exiger qu'un programme couvert par le gauche d'auteur ne puisse pas être inclus dans une version plus grande sans que cette dernière ne soit également couverte par le gauche d'auteur.

L'implémentation spécifique du gauche d'auteur que nous avons utilisée pour la plupart des logiciels GNU fut la GNU General Public License (licence publique générale de GNU), ou GNU GPL en abrégé. Nous disposons d'autres types de gauche d'auteur pour des circonstances particulières. Les manuels du projet GNU sont eux aussi couverts par le gauche d'auteur, mais en utilisent une version très simplifiée, car il n'est pas nécessaire de faire appel à toute la complexité de la GNU GPL dans le cadre de manuels.

5.11 La Free Software Foundation, ou fondation du logiciel libre

Emacs attirant de plus en plus l'attention, le projet GNU comptait un nombre croissant de participants, et nous avons décidé qu'il était temps de repartir à la chasse aux fonds. En 1985, nous avons donc créé la fondation du logiciel libre (FSF), une association à but non lucratif, exemptée d'impôts, pour le développement de logiciel libre. La FSF a récupéré le marché de la distribution de logiciel libre sur bandes, auxquelles elle ajouta ensuite d'autres logiciels libres (GNU ou non), et par la vente de manuels libres.

La FSF accepte les dons, mais la plus grande partie de ses recettes est toujours provenue des ventes — de copies de logiciel libre ou d'autres services associés. De nos jours, elle vend des CD-ROM de code source, des CD-ROM de binaires, des manuels de qualité (tout cela, en autorisant la redistribution et les modifications), et des distributions « Deluxe » (dans lesquelles nous construisons tous les logiciels pour la plate-forme de votre choix).

Les employés de la fondation du logiciel libre ont écrit et maintenu un grand nombre de paquetages logiciels du projet GNU, en particulier la bibliothèque du langage C et l'interpréteur de commandes. La bibliothèque du langage C est ce qu'utilise tout programme fonctionnant sur un système GNU/Linux pour communiquer avec Linux. Elle a été développée par Roland McGrath, membre de l'équipe de la fondation du logiciel libre. L'interpréteur de commandes employé

sur la plupart des systèmes GNU/Linux est BASH, le Bourne-Again Shell [9], qui a été développé par Brian Fox, employé de la FSF.

Nous avons financé le développement de ces programmes car le projet GNU ne se limitait pas aux outils ou à un environnement de développement. Notre but était la mise en place d'un système d'exploitation complet, et de tels programmes étaient nécessaires pour l'atteindre.

5.12 Assistance technique au logiciel libre

La philosophie du logiciel libre rejette une pratique spécifique, très répandue dans l'industrie du logiciel, mais elle ne s'oppose pas au monde des affaires. Quand des entreprises respectent la liberté des utilisateurs, nous leur souhaitons de réussir.

La vente de copies d'Emacs est une forme d'affaires fondées sur du logiciel libre. Quand la FSF a récupéré ce marché, j'ai dû chercher une autre solution pour gagner ma vie. Je l'ai trouvée sous la forme de vente de services associés au logiciel libre que j'avais développé. Cela consistait à enseigner des thèmes tels que la programmation de GNU Emacs et la personnalisation de GCC, et à développer du logiciel, principalement en portant GCC sur de nouvelles plates-formes.

De nos jours, chacune de ces activités lucratives fondées sur le logiciel libre est proposée par de nombreuses sociétés. Certaines distribuent des compilations de logiciel libre sur CD-ROM ; d'autres vendent de l'assistance technique en répondant à des questions d'utilisateurs, en corrigeant des bogues, et en insérant de nouvelles fonctionnalités majeures. On commence même à observer des sociétés de logiciel libre fondées sur la mise sur le marché de nouveaux logiciels libres.

Prenez garde, toutefois — certaines des sociétés qui s'associent à la dénomination « open source » [10] fondent en réalité leur activité sur du logiciel propriétaire, qui interagit avec du logiciel libre. Ce ne sont pas des sociétés de logiciel libre, ce sont des sociétés de logiciel propriétaire dont les produits détournent les utilisateurs de leur liberté. Elles appellent cela de la « valeur ajoutée », ce qui reflète quelles valeurs elles souhaitent nous voir adopter : préférer la facilité à la liberté. Si nous faisons passer la liberté au premier plan, il nous faut leur donner le nom de produits à « liberté soustraite ».

5.13 Objectifs techniques

L'objectif principal du projet GNU était le logiciel libre. Même si GNU ne jouissait d'aucun avantage technique sur Unix™, il disposerait d'un avantage

9. « Bourne-Again Shell » est un clin d'œil au nom « Bourne Shell », qui était l'interpréteur de commandes habituel sur Unix™ (N.d.T. : le mot anglais bash a le sens de « coup, choc » et la signification de cet acronyme est double ; c'est à la fois une nouvelle version de l'interpréteur de commandes Bourne, et une allusion aux chrétiens qui se sont sentis renaître dans cette religion, et qu'aux États-Unis d'Amérique on qualifie de *born again Christians*).

10. littéralement, « [logiciel dont le] code source est ouvert ». C'est une périphrase lourde et inélégante en français, mais qui résout en anglais l'ambiguïté discutée plus haut, bien que l'auteur rejette cette solution, pour des raisons expliquées à la fin de cet article. Il s'agit ici de sociétés qui font peu de cas du logiciel libre et choisissent un slogan calculé pour s'attirer les faveurs du public.

social, en autorisant les utilisateurs à coopérer, et d'un avantage éthique, en respectant la liberté de l'utilisateur.

Mais il était naturel d'appliquer à ce travail les standards bien connus du développement logiciel de qualité — en utilisant par exemple des structures de données allouées dynamiquement pour éviter de mettre en place des limites fixées arbitrairement, et en gérant tous les caractères possibles encodables sur 8 bits, partout où cela avait un sens.

De plus, nous rejetions l'accent mis par Unix™ sur les petites quantités de mémoire, en décidant de ne pas nous occuper des architectures 16 bits (il était clair que les architectures 32 bits seraient la norme au moment de la finalisation du système GNU), et en ne faisant aucun effort pour réduire la consommation mémoire en deçà d'un méga-octet. Dans les programmes pour lesquels il n'était pas crucial de manipuler des fichiers de tailles importantes, nous encouragions les programmeurs à lire le fichier en entrée, d'une traite, en mémoire, et d'analyser ensuite son contenu sans plus se préoccuper des entrées/sorties.

Ces décisions ont rendu de nombreux programmes du projet GNU supérieurs à leurs équivalents sous Unix™ en termes de fiabilité et de vitesse d'exécution.

5.14 Les ordinateurs offerts

La réputation du projet GNU croissant, on nous offrait des machines sous Unix™ pour nous aider à le mener à bien. Elles nous furent bien utiles, car le meilleur moyen de développer les composants de GNU était de travailler sur un système Unix™, dont on remplaçait les composants un par un. Mais cela a posé un problème éthique : était-il correct ou non, pour nous, de posséder des copies d'Unix™ ?

Unix™ était (et demeure) du logiciel propriétaire, et la philosophie du projet GNU nous demandait de ne pas utiliser de logiciels propriétaire. Mais, en appliquant le même raisonnement que celui qui conclut qu'il est légitime de faire usage de violence en situation de légitime défense, j'ai conclu qu'il était légitime d'utiliser un paquetage propriétaire quand cela était crucial pour développer une solution de remplacement libre, qui en aiderait d'autres à se passer de ce même paquetage propriétaire.

Mais ce mal avait beau être justifiable, il n'en restait pas moins un mal. De nos jours, nous ne possédons plus aucune copie d'Unix™, car nous les avons toutes remplacées par des systèmes d'exploitation libres. Quand nous ne parvenions pas à substituer au système d'exploitation d'une machine un système libre, nous remplacions la machine.

5.15 La GNU Task List, ou liste des tâches du projet GNU

Le projet GNU suivant son cours, on trouvait ou développait un nombre croissant de composants du système, et il est finalement devenu utile de faire la liste des parties manquantes. Nous l'avons utilisée pour recruter des développeurs afin d'écrire ces dernières. Cette liste a pris le nom de GNU task list. En plus des composants manquants d'Unix™, nous y avons listé plusieurs autres

projets utiles, de logiciel et de documentation, que nous jugions nécessaires au sein d'un système réellement complet.

De nos jours, on ne trouve presque plus aucun composant d'Unix™ dans la liste des tâches du projet GNU — ces travaux tous ont été menés à bien, si on néglige certains composants non essentiels. Mais la liste est pleine de projets qu'on pourrait qualifier d'« applications ». Tout programme qui fait envie à une classe non restreinte d'utilisateurs constituerait un ajout utile à un système d'exploitation.

On trouve même des jeux dans la liste des tâches — et c'est le cas depuis le commencement. Unix™ proposait des jeux, ce devait naturellement être également le cas de GNU. Mais il n'était pas nécessaire d'être compatible en matière de jeux, aussi n'avons-nous pas suivi la liste des jeux d'Unix™. Nous avons plutôt listé un spectre de divers types de jeux qui plairont vraisemblablement aux utilisateurs.

5.16 La GNU Library GPL, ou licence publique générale de GNU pour les bibliothèques

La bibliothèque du langage C du projet GNU fait appel à un gauche d'auteur particulier, appelé la GNU Library General Public License (licence publique générale de GNU pour les bibliothèques, ou GNU LGPL), qui autorise la liaison de logiciel propriétaire avec la bibliothèque. Pourquoi une telle exception ?

Ce n'est pas une question de principe ; aucun principe ne dicte que les logiciels propriétaires ont le droit de contenir notre code (pourquoi contribuer à un projet qui affirme refuser de partager avec nous ?). L'utilisation de la LGPL dans le cadre de la bibliothèque du langage C, ou de toute autre bibliothèque, est un choix stratégique.

La bibliothèque du langage C joue un rôle générique ; tout système propriétaire, tout compilateur, dispose d'une bibliothèque du langage C. C'est pourquoi limiter l'utilisation de notre bibliothèque du langage C au logiciel libre n'aurait donné aucun avantage au logiciel libre — cela n'aurait eu pour effet que de décourager l'utilisation de notre bibliothèque.

Il existe une exception à cette règle : sur un système GNU (et GNU/Linux est l'un de ces systèmes), la bibliothèque du langage C de GNU est la seule disponible. Aussi, ses conditions de distribution déterminent s'il est possible de compiler un programme propriétaire sur le système GNU. Il n'existe aucune raison éthique d'autoriser des applications propriétaires sur le système GNU, mais d'un point de vue stratégique, il semble que les interdire découragerait plus l'utilisation d'un système GNU que cela n'encouragerait le développement d'applications libres.

C'est pourquoi l'utilisation de la GPL pour les bibliothèques (ou LGPL) est une bonne stratégie dans le cadre de la bibliothèque du langage C. En ce qui concerne les autres bibliothèques, il faut prendre la décision stratégique au cas par cas. Quand une bibliothèque remplit une tâche particulière qui peut faciliter l'écriture de certains types de programmes, la distribuer sous les conditions de la GPL, en limitant son utilisation aux programmes libres, est une manière d'aider les développeurs de logiciels libres et de leur accorder un avantage à l'encontre du logiciel propriétaire.

Considérons GNU Readline, une bibliothèque développée dans le but de fournir une édition de ligne de commande pour l'interpréteur de commandes BASH. Cette bibliothèque est distribuée sous la licence publique générale ordinaire de GNU, et non pas sous la LGPL. Cela a probablement pour effet de réduire l'utilisation de la bibliothèque Readline, mais cela n'induit aucune perte en ce qui nous concerne. Pendant ce temps, on compte au moins une application utile qui a été libérée, uniquement dans le but de pouvoir utiliser la bibliothèque Readline, et c'est là un gain réel pour la communauté.

Les développeurs de logiciel propriétaire jouissent des avantages que leur confère l'argent ; les développeurs de logiciel libre doivent compenser cela en s'épaulant les uns les autres. J'espère qu'un jour nous disposerons de toute une collection de bibliothèques couvertes par la GPL, et pour lesquelles il n'existera pas d'équivalent dans le monde du logiciel propriétaire. Nous disposerons ainsi de modules utiles, utilisables en tant que blocs de construction de nouveaux logiciels libres, et apportant un avantage considérable à la continuation du développement du logiciel libre.

5.17 Gratter là où ça démange ?

Eric Raymond affirme que « Tout bon logiciel commence par gratter un développeur là où ça le démange. ». Cela se produit peut-être, parfois, mais de nombreux composants essentiels du logiciel GNU ont été développés dans le but de disposer d'un système d'exploitation libre complet. Ils ont été inspirés par une vision et un projet à long terme, pas par un coup de tête.

Nous avons par exemple développé la bibliothèque du langage C de GNU car un système de type Unix™ a besoin d'une bibliothèque du langage C, nous avons développé le Bourne-Again Shell (BASH) car un système de type Unix™ a besoin d'un interpréteur de commandes, et nous avons développé GNU tar car un système de type Unix™ a besoin d'un programme d'archivage. Il en va de même pour les programmes que j'ai développé, à savoir le compilateur C de GNU, GNU Emacs, GDB, et GNU Make.

Certains programmes du projet GNU ont été développés pour répondre aux menaces qui pesaient sur notre liberté. C'est ainsi que nous avons développé gzip en remplacement du programme Compress, que la communauté avait perdu suite aux brevets logiciels déposés sur LZW. Nous avons trouvé des gens pour développer LessTif, et plus récemment nous avons démarré les projets GNOME et Harmony, en réponse aux problème posés par certaines bibliothèques propriétaires (lire ci-après). Nous sommes en train de développer le GNU Privacy Guard (le gardien de l'intimité de GNU, ou GPG) pour remplacer un logiciel de chiffrement populaire mais pas libre, car les utilisateurs ne devraient pas devoir choisir entre la préservation de leur intimité et la préservation de leur liberté.

Bien sûr, les gens qui écrivent ces programmes se sont intéressés à ce travail, et de nombreux contributeurs ont ajouté de nouvelles fonctionnalités car elles comblaient leurs besoins ou les intéressaient. Mais ce n'est pas là la raison première de ces programmes.

5.18 Des développements inattendus

Au commencement du projet GNU, j'ai imaginé que nous développerions le système GNU dans sa globalité avant de le publier. Les choses se sont passées différemment.

Puisque chaque composant du système GNU était implémenté sur un système Unix™, chaque composant pouvait fonctionner sur des systèmes Unix™, bien avant que le système GNU ne soit disponible dans sa globalité. Certains de ces programmes sont devenus populaires, et leurs utilisateurs ont commencé à travailler sur des extensions et des ports — vers les diverses versions d'Unix™, incompatibles entre elles, et parfois, sur d'autres systèmes encore.

Ce processus a rendu ces programmes bien plus complets, et a drainé des fonds et des participants vers le projet GNU. Mais il a probablement eu également pour effet de retarder de plusieurs années la mise au point d'un système en état de fonctionnement, puisque les développeurs du projet GNU passaient leur temps à s'occuper de ces ports et à proposer des nouvelles fonctionnalités aux composants existants, plutôt que de continuer à développer peu à peu les composants manquants.

5.19 Le GNU Hurd

En 1990, le système GNU était presque terminé ; le seul composant principal qui manquait encore à l'appel était le noyau. Nous avions décidé d'implémenter le noyau sous la forme d'une série de processus serveurs qui fonctionneraient au-dessus de Mach. Mach est un micro-noyau développé à l'université Carnegie-Mellon puis à l'université de l'Utah ; le GNU Hurd est une série de serveurs (ou une « horde de gnous ») qui fonctionnent au-dessus de Mach, et remplissent les diverses fonctions d'un noyau Unix™. Le développement a été retardé car nous attendions que Mach soit publié sous forme de logiciel libre, comme cela avait été promis.

L'une des raisons qui ont dicté ce choix était d'éviter ce qui semblait être la partie la plus difficile du travail : déboguer un programme de noyau sans disposer pour cela d'un débogueur au niveau du code source. Ce travail avait déjà été fait, dans Mach, et nous pensions déboguer les serveurs du Hurd en tant que programmes utilisateur, à l'aide de GDB. Mais cela prit beaucoup de temps, et les serveurs à plusieurs processus légers, qui s'envoyaient des messages les uns aux autres, se sont révélés très difficiles à déboguer. La consolidation du Hurd s'est étalée sur plusieurs années.

5.20 Alix

Á l'origine, le noyau du système GNU n'était pas censé s'appeler Hurd. Son premier nom était Alix — du nom de celle qui à l'époque était l'objet de ma flamme. Administratrice de systèmes Unix™, elle avait fait remarquer que son prénom ressemblait aux noms typiques des versions de systèmes Unix™ ; elle s'en était ouverte auprès d'amis en plaisantant : « Il faudrait baptiser un noyau de mon nom. » Je suis resté coi, mais ai décidé de lui faire la surprise d'appeler Alix le noyau du système GNU.

Mais les choses ont changé. Michael Bushnell (maintenant, il s'agit de Thomas), le développeur principal du noyau, préférait le nom Hurd, et a confiné le nom Alix à une certaine partie du noyau — la partie qui se chargeait d'intercepter les appels système et de les gérer en envoyant des messages aux serveurs du Hurd.

Finalement, Alix et moi mîmes fin à notre relation, et elle a changé de nom ; de manière indépendante, le concept du Hurd avait évolué de telle sorte que ce serait la bibliothèque du langage C qui enverrait directement des messages aux serveurs, ce qui a fait disparaître le composant Alix du projet.

Mais avant que ces choses ne se produisissent, un de ses amis avait remarqué le nom Alix dans le code source du Hurd, et s'en était ouvert auprès d'elle. Finalement, ce nom avait rempli son office.

5.21 Linux et GNU/Linux

Le GNU Hurd n'est pas encore utilisable de manière intensive. Heureusement, on dispose d'un autre noyau. En 1991, Linus Torvalds a développé un noyau compatible avec Unix™ et lui a donné le nom de Linux. Aux alentours de 1992, la jonction de Linux et du système GNU, qui était presque complet, a fourni un système d'exploitation libre et complet (ce travail de jonction était lui-même, bien sûr, considérable). C'est grâce à Linux qu'on peut désormais employer une version du système GNU.

On appelle cette version du système « GNU/Linux » pour signaler qu'il est composé du système GNU et du noyau Linux.

5.22 Les défis à venir

Nous avons fait la preuve de notre capacité à développer un large spectre de logiciel libre. Cela ne signifie pas que nous sommes invincibles et que rien ne peut nous arrêter. Certains défis rendent incertain l'avenir du logiciel libre ; et il faudra des efforts et une endurance soutenus pour les relever, pendant parfois plusieurs années. Il faudra montrer le genre de détermination dont les gens font preuve quand ils accordent de la valeur à leur liberté et qu'ils ne laisseront personne la leur voler.

Les cinq sections suivantes discutent de ces défis.

5.22.1 Le matériel secret

Les fabricants de matériel tendent de plus en plus à garder leurs spécifications secrètes. Cela rend plus difficile l'écriture de pilotes de périphériques libres afin de permettre à Linux et au projet XFree86 de reconnaître de nouveaux matériels. Nous disposons aujourd'hui de systèmes entièrement libres, mais cela pourrait ne plus être le cas dans l'avenir, si nous ne pouvons plus proposer des pilotes pour les ordinateurs de demain.

On peut résoudre ce problème de deux manières. Les programmeurs peuvent analyser l'ensemble afin de deviner comment prendre en compte le matériel. Les autres peuvent choisir le matériel qui est reconnu par du logiciel libre ; plus

nous serons nombreux, plus la politique de garder les spécifications secrètes sera vouée à l'échec.

L'ingénierie à l'envers est un travail conséquent ; disposerons-nous de programmeurs suffisamment déterminés pour le prendre en main ? Oui — si nous avons construit un sentiment puissant selon lequel le logiciel libre est une question de principe, et que les pilotes non libres sont inacceptables. Et serons-nous nombreux à dépenser un peu plus d'argent, ou à passer un peu de temps, pour que nous puissions utiliser des pilotes libres ? Oui — si la détermination afférente à la liberté est largement répandue.

5.22.2 Les bibliothèques non libres

Une bibliothèque non libre qui fonctionne sur des systèmes d'exploitation libres se comporte comme un piège vis-à-vis des développeurs de logiciel libre. Les fonctionnalités attrayantes de cette bibliothèque sont l'appât ; si vous utilisez la bibliothèque, vous tombez dans le piège, car votre programme ne peut pas être utilisé de manière utile au sein d'un système d'exploitation libre (pour être strict, on pourrait y inclure le programme, mais on ne pourrait pas l'exécuter en l'absence de la bibliothèque incriminée). Pire encore, si un programme qui utilise une bibliothèque propriétaire devient populaire, il peut attirer d'autres programmeurs peu soupçonneux dans le piège.

Ce problème s'est posé pour la première fois avec la boîte à outils Motif, dans les années 80. Même s'il n'existait pas encore de systèmes d'exploitation libres, il était limpide que Motif leur causerait des problèmes, plus tard. Le projet GNU a réagi de deux manières : en demandant aux projets de logiciel libre de rendre l'utilisation de Motif facultative en privilégiant les gadgets de la boîte à outils X, libre, et en recherchant un volontaire pour écrire une solution de remplacement libre à Motif. Ce travail prit de nombreuses années ; LessTif, développé par les Hungry Programmers (les « Programmeurs affamés »), n'est devenu suffisamment étendu pour faire fonctionner la plupart des applications utilisant Motif qu'en 1997.

De 1996 à 1998, une compilation conséquente de logiciel libre, le bureau KDE, a fait usage d'une autre bibliothèque non libre de boîte à outils pour l'interface graphique utilisateur, appelée Qt.

Les systèmes GNU/Linux libres ne pouvaient pas utiliser KDE, car nous ne pouvions pas utiliser la bibliothèque. Cependant, certains distributeurs commerciaux de systèmes GNU/Linux n'ont pas été assez stricts pour coller au logiciel libre et ont ajouté KDE dans leurs systèmes — produisant un système disposant d'un plus grand nombre de fonctionnalités, mais souffrant d'une liberté réduite. Le groupe KDE encourageait activement un plus grand nombre de programmeurs à utiliser la bibliothèque Qt, et des millions de « nouveaux utilisateurs de Linux » n'ont jamais eu connaissance du fait que tout ceci posait un problème. La situation était sinistre.

La communauté du logiciel libre a répondu à ce problème de deux manières : GNOME et Harmony.

GNOME, le GNU Network Object Model Environment (environnement de GNU de modèle d'objets pour le réseau), est le projet de bureau de GNU. Démarré en 1997 par Miguel de Icaza, et développé avec l'aide de la société Red Hat Software™, GNOME avait pour but de fournir des fonctionnalités de bureau similaires, en utilisant exclusivement du logiciel libre. Il jouit aussi

d'avantages techniques, comme le fait de collaborer avec toute une variété de langages, et de ne pas de se limiter au C++. Mais son objectif principal est la liberté : ne pas imposer l'utilisation du moindre logiciel non libre.

Harmony est une bibliothèque compatible de remplacement, conçue pour permettre l'utilisation des logiciels de KDE sans faire appel à Qt.

En novembre 1998, les développeurs de Qt ont annoncé une modification de leur licence qui, quand elle sera effective, fera de Qt un logiciel libre. On ne peut pas en être sûr, mais je pense que cette décision est en partie imputable à la réponse ferme qu'a faite la communauté au problème que Qt posait quand il n'était pas libre (la nouvelle licence n'est pas pratique ni équitable, aussi demeure-t-il préférable d'éviter d'utiliser Qt).

Comment répondrons-nous à la prochaine bibliothèque non libre mais alléchante ? La communauté comprendra-t-elle dans son entier la nécessité de ne pas tomber dans le piège ? Ou serons-nous nombreux à préférer la facilité à la liberté, et à produire un autre problème majeur ? Notre avenir dépend de notre philosophie.

5.22.3 Les brevets sur les logiciels

La pire menace provient des brevets sur les logiciels, susceptibles de placer des algorithmes et des fonctionnalités hors de portée des logiciels libres pendant une période qui peut atteindre vingt ans. Les brevets sur l'algorithme de compression LZW ont été déposés en 1983, et nous ne pouvons toujours pas diffuser des logiciels libres qui produisent des images au format GIF correctement compressées. En 1998, la menace d'une poursuite pour cause de violation de brevets a mis fin à la distribution d'un programme libre qui produisait des données sonores compressées au format MP3.

Il existe plusieurs manières de répondre au problème des brevets : on peut rechercher des preuves qui invalident un brevet, et on peut rechercher d'autres solutions pour remplir une tâche. Mais chacune de ces méthodes ne fonctionne que dans certains cas ; quand les deux échouent, il se peut qu'un brevet empêche le logiciel libre de disposer de fonctionnalités souhaitées par les utilisateurs. Que ferons-nous dans ce genre de situation ?

Ceux d'entre nous qui prêtent de la valeur au logiciel libre par amour de la liberté continueront à utiliser du logiciel libre dans tous les cas. On pourra travailler sans utiliser de fonctionnalités protégées par des brevets. Mais ceux d'entre nous qui prêtent de la valeur au logiciel libre car ils s'attendent à trouver là des logiciels techniquement supérieurs sont susceptibles de critiquer l'idée même du logiciel libre quand un brevet l'empêchera de progresser plus avant. Ainsi, même s'il est utile de discuter de l'efficacité, dans la pratique, du modèle de développement de type « cathédrale », et de la fiabilité et de la puissance de certains logiciels libres, il ne faut pas s'en tenir là. Il nous faut parler de liberté et de principes.

5.22.4 La documentation libre

Il ne faut pas chercher les lacunes les plus graves de nos systèmes d'exploitations libres dans le logiciel — c'est l'absence de manuels libres corrects qu'on puisse inclure dans nos systèmes qui se fait le plus cruellement sentir. La documentation est essentielle dans tout paquetage logiciel ; quand un paquetage

logiciel important ne dispose pas d'un bon manuel libre, il s'agit d'un manque crucial. On en compte de nombreux aujourd'hui.

La documentation libre, tout comme le logiciel libre, est une question de liberté, pas de prix[11]. La raison d'être d'un manuel libre est très proche de celle d'un logiciel libre : il s'agit d'offrir certaines libertés à tous les utilisateurs. Il faut autoriser la redistribution (y compris la vente commerciale), en ligne et sur papier, de telle sorte que le manuel puisse accompagner toute copie du programme.

Il est également crucial d'autoriser les modifications. En règle générale, je ne pense pas qu'il soit essentiel d'autoriser tout un chacun à modifier toutes sortes d'articles et de livres. Je ne pense pas, par exemple, que vous ou moi soyons tenus de donner la permission de modifier des textes comme le présent article, qui expose nos actions et nos idées.

Mais il existe une raison particulière, pour laquelle il est crucial de disposer de la liberté de modifier la documentation afférente au logiciel libre. Quand on jouit de son droit de modifier le logiciel, et d'ajouter des fonctionnalités ou de modifier les fonctionnalités présentes, le programmeur consciencieux mettra immédiatement à jour le manuel — afin de fournir une documentation précise et utilisable aux côtés du programme modifié. Un manuel qui n'autorise pas les programmeurs à être consciencieux et à terminer leur travail, ne remplit pas les besoins de notre communauté.

Il est acceptable d'apposer certaines limites sur la manière dont les modifications sont faites. Il est par exemple envisageable d'exiger de préserver la notice de copyright de l'auteur original, les conditions de distribution, ou la liste des auteurs. D'exiger que les versions modifiées contiennent une notice qui stipule qu'elles ont été modifiées, et même d'interdire de modifier ou d'ôter des sections entières, pourvu que ces sections ne traitent pas de considérations techniques, ne pose pas non plus de problèmes, car cela n'interdit pas au programmeur consciencieux d'adapter le manuel afin qu'il corresponde au programme modifié par ses soins. En d'autres termes, cela n'empêche la communauté du logiciel libre d'utiliser pleinement le manuel.

En revanche, il faut autoriser la modification des portions techniques du manuel, et la distribution du résultat de ces modifications par tous les médias habituels, à travers tous les canaux habituels ; sans quoi, les restrictions font obstruction à la communauté, le manuel n'est pas libre, et il nous en faut un autre.

Les développeurs de logiciels libres seront-ils déterminés, auront-ils conscience du fait qu'il est nécessaire de produire tout un spectre de manuels libres ? Une fois de plus, notre avenir dépend de notre philosophie.

5.22.5 Il nous faut faire l'apologie de la liberté

On estime aujourd'hui à dix millions le nombre d'utilisateurs de systèmes GNU/Linux et Red Hat Linux™ de par le monde. Le logiciel libre propose tant d'avantages pratiques que les utilisateurs s'y ruent pour des raisons purement pratiques.

Cet état de fait a des conséquences heureuses, qui n'échapperont à personne : on voit plus de développeurs intéressés par la production de logiciels libres,

11. ici encore, les anglais sont victimes de l'absence de mot adéquat pour signifier « libre ».

les entreprises de logiciels libres comptent plus de clients, et il est plus facile d'encourager les sociétés à développer des logiciels libres commerciaux, plutôt que des produits logiciels propriétaires.

Mais l'intérêt pour le logiciel libre croît plus vite que la prise de conscience de la philosophie sur laquelle il se fonde, et cela provoque des problèmes. Notre capacité à relever les défis et à répondre aux menaces évoqués plus haut dépend de notre volonté à défendre chèrement notre liberté. Pour nous assurer que notre communauté partage cette volonté, il nous faut répandre ces idées auprès des nouveaux utilisateurs au fur et à mesure qu'ils rejoignent notre communauté.

Mais nous négligeons ce travail ; on dépense bien plus d'efforts pour attirer de nouveaux utilisateurs dans notre communauté qu'on n'en dépense pour leur enseigner l'éducation civique qui lui est attachée. Ces deux efforts sont nécessaires, et il nous faut les équilibrer.

5.23 « Open Source »

En 1998, il est devenu plus difficile de sensibiliser les nouveaux utilisateurs à la notion de liberté dans le logiciel, quand une portion de notre communauté a choisi d'arrêter d'utiliser le terme « Free Software » pour lui préférer la dénomination « Open Source software » [12].

Certains de ceux qui ont choisi ce nouveau nom avaient en tête de mettre fin à la confusion souvent constatée entre les mots « free » et « gratuit » — ce qui est un objectif valable. D'autres, au contraire, avaient pour objectif de laisser de côté le principe qui a depuis toujours motivé le mouvement du logiciel libre et le projet GNU, afin de cibler les cadres et les utilisateurs professionnels, dont beaucoup ont une idéologie où la liberté, la communauté, et les principes, cèdent le pas aux profits. Ainsi, la rhétorique de l'« Open Source » met l'accent sur le potentiel pour faire du logiciel puissant et de grande qualité, mais occulte délibérément les idées de liberté, de communauté, et de principes.

Les magazines « Linux » illustrent clairement cet exemple — ils sont bourrés de publicités pour des logiciels propriétaires qui fonctionnent sur le système GNU/Linux. Quand le prochain Motif ou Qt poindra, ces magazines mettront-ils les programmeurs en garde en leur demandant de s'en éloigner, ou passeront-ils des publicités pour ces produits ?

La communauté a beaucoup à gagner de la participation des entreprises ; toutes choses étant égales par ailleurs, cette contribution est utile. Mais sacrifier à cette aide les discours traitant de liberté et de principes peut avoir des conséquences désastreuses ; cela déséquilibre encore plus la situation précédente, où on voit que l'éducation civique des nouveaux utilisateurs s'avère difficile lorsqu'ils affluent.

Les termes « Free Software » et « Open Source » décrivent tous deux plus ou moins la même catégorie de logiciels, mais correspondent à des conceptions différentes du logiciel et des valeurs qui lui sont associées. Le projet GNU continue d'utiliser le terme « Free Software » pour exprimer l'idée que la liberté est plus importante que la seule technique.

12. encore et toujours cette ambiguïté de la langue anglaise. « software » signifie « logiciel ». « free » signifie à la fois « libre », sens qui est pertinent ici, et « gratuit », qualité qui n'est qu'un effet de bord des logiciels libres. « open source » signifie « dont le code source est ouvert ».

5.24 Jetez-vous à l'eau !

La philosophie de Yoda (« il ne faut pas *essayer* ») est attirante, mais elle ne s'applique pas à moi. J'ai effectué la plupart de mes travaux sans savoir si j'étais capable de les mener à bien, et sans savoir si ces derniers, une fois menés à bien, suffiraient aux buts que je leur avais fixés. Mais j'ai tenté ma chance, car il n'y avait personne d'autre que moi entre l'ennemi et ma cité. À ma grande surprise, j'ai parfois réussi.

J'ai parfois échoué ; certaines de mes cités sont tombées. Je trouvais alors une autre cité menacée, et je me préparais pour une nouvelle bataille. Avec le temps, j'ai appris à reconnaître les menaces et à m'interposer entre ces dernières et ma cité, en appelant mes amis hackers à la rescousse.

Maintenant, il arrive souvent que je ne sois pas seul. C'est pour moi un soulagement et une joie de constater que tout un régiment de hackers se mobilise pour faire front, et je réalise qu'il se peut que cette cité survive — pour le moment. Mais les dangers grandissent chaque année, et maintenant la société Microsoft™ a explicitement pris notre communauté dans son collimateur. L'avenir de la liberté n'est pas un fait acquis. Ne le considérez pas comme tel ! Si vous souhaitez conserver votre liberté, il vous faut vous préparer à la défendre.

5.25 Informations légales

Chapitre 6

L'avenir de Cygnus Solutions™, Récit d'un entrepreneur

Créée en 1989, Cygnus Solutions™ fut la toute première, et selon un rapport du magazine Forbes en août 1998, c'est aujourd'hui la plus importante entreprise fondée sur les logiciels libres. Son produit phare, le kit de développement GNUPro, est le leader du marché des compilateurs et des débogueurs pour logiciels embarqués. Cygnus compte parmi ses clients les plus grands fabricants de microprocesseurs, et des leaders dans le domaine de l'électronique grand public, de l'Internet, des télécommunications, de l'automatisation, des réseaux, de l'aérospatiale ou de l'automobile. Son siège se trouve à Sunnyvale, en Californie, elle a des bureaux à Atlanta, Cambridge, Tokyo et Toronto, et des employés en divers endroits allant de l'Australie à l'Oregon. Cygnus est la plus grande entreprise à capitaux privés dans l'industrie du logiciel embarqué, plus importante que deux entreprises à capitaux publics et sur le points de dépasser la troisième entreprise par ordre d'importance. Avec une croissance du chiffre d'affaires de 65 depuis 1992, Cygnus a figuré dans le Top 100 de la croissance du CA du San Jose Business Journal's. Elle figure maintenant dans la Software 500 list (tous secteurs confondus).

Dans cet article, je décrirai le modèle du logiciel libre qui est la clé de notre succès, et la manière dont nous continuons à le revoir et à l'améliorer afin de progresser.

Ce n'est que le 13 novembre 1989 que nous avons reçu la lettre du California Department of Corporations nous informant que notre demande était approuvée, que nous pouvions déposer notre capital de 6000 et commencer notre activité sous le nom de « Cygnus Support ». C'était l'aboutissement d'un projet né plus de 2 ans auparavant, et le début d'une aventure qui continue encore aujourd'hui, presque 10 ans plus tard.

Cette vision a commencé de façon relativement innocente. Mon père m'a dit un jour : « Quand tu lis un livre, lis-le de la première à la dernière page ». Comme tous les conseils paternels, celui-ci ne fut appliqué que lorsque j'en avais envie. En 1987, trouvant mon travail de plus en plus ennuyeux, je commençai à m'intéresser aux logiciels GNU, et je décidai de lire le *GNU Emacs Manual*, publié à compte d'auteur par Richard Stallman, de la première à la dernière page (aucun éditeur n'avait voulu publier ce livre, car ses lecteurs étaient non seulement autorisés mais encore encouragés à en faire des copies gratuites. De fait, même aujourd'hui, c'est un concept toujours difficile à expliquer aux éditeurs).

Emacs est un programme fascinant. Bien plus qu'un éditeur de texte ; il avait été étendu pour permettre de lire son courrier électronique, les forums Usenet, exploiter un shell, lancer des compilations et déboguer les programmes résultants ; il donne même accès à l'interprète Lisp™ sur lequel il repose. Des utilisateurs inventifs (ou, ce qui revient au même, des hackers qui s'ennuyaient) avaient ajouté des fonctions saugrenues comme le mode doctor de psychanalyse automatique inspiré du programme ELIZA de John Mc Carthy, ou encore le mode dissociated-press qui réarrange le texte de façon bizarre et très amusante. Il y a même un programme qui dessine et anime la solution du problème des tours de Hanoï (sur un écran de texte !). C'étaient cette profondeur et cette richesse qui m'ont donné envie d'en savoir plus, de lire le manuel et le code source d'Emacs.

Le dernier chapitre de ce livre, le « manifeste GNU », est une réponse personnelle de l'auteur à la question qui m'a travaillé durant toute ma lecture, à savoir : pourquoi un programme aussi bon est-il disponible gratuitement avec ses sources et librement redistribuable, en un mot libre ? Stallman répond à la

question plus générale suivante :

> Pourquoi il faut que j'écrive GNU ?
>
> Je considère que la règle d'or est que si j'apprécie un programme je dois le partager avec d'autres qui l'apprécient. Les éditeurs de logiciels veulent nous diviser pour nous conquérir, en persuadant chaque utilisateur de ne pas vouloir partager avec les autres. Je refuse ainsi de briser la solidarité avec les autres utilisateurs.

On ne peut pas citer tout le « Manifeste » ici. On dira simplement que s'il ressemble en surface à un pamphlet socialiste, j'ai voulu y voir autre chose. En fait j'y ai vu un business plan. L'idée est simple : le logiciel libre unifie les efforts des programmeurs du monde entier, et des entreprises proposant des services (personnalisation, améliorations, corrections, assistance technique) pour ces logiciels profiteraient des économies d'échelle et du fort pouvoir d'attraction de cette nouvelle espèce de logiciel.

Emacs n'était pas le seul programme époustouflant de la *Free Software Foundation*. Il y avait le débogueur GNU (gdb), que Stallman a dû écrire parce que les débogueurs de DEC (maintenant filiale de Compaq) et de Sun n'étaient pas assez puissants pour un programme aussi complexe qu'Emacs. Non seulement ce programme pouvait tolérer une charge importante, mais il le faisait avec élégance : ses commandes et ses extensions étaient vraiment adaptées aux programmeurs. Et comme c'était un logiciel libre, des programmeurs n'ont pas tardé à ajouter d'autres extensions qui l'ont rendu encore plus puissant. Les logiciels propriétaires n'offraient pas ce type d'extensibilité.

Mais la vraie bombe tomba en juin 1987, lorsque Stallman publia la version 1.0 du compilateur C de GNU, gcc. Je l'ai immédiatement téléchargé et j'ai utilisé tous les trucs du manuel de gdb et d'Emacs pour maîtriser rapidement ses 110000 lignes de code. La première version du compilateur de Stallman était compatible avec deux plates-formes : le vénérable VAX et les nouvelles stations Sun3. Il générait un code qui tenait la dragée haute aux compilateurs de DEC et Sun. J'ai porté gcc sur le nouveau processeur 32032 de National Semiconductor. Le résultat était 20 fois plus rapide que le compilateur propriétaire fourni par National. Après deux semaines de bidouilles supplémentaires, j'ai porté la différence à 40 (on a souvent dit que la puce de National n'a pas connu le succès parce qu'elle ne délivrait pas 1Mips et ne concurrençait donc pas le 68020 de Motorola. Or les benchmarks faits avec des applications lui donnaient juste 0,75 Mips à sa sortie. Mais 140 x 0,75 Mips = 1,05 Mips. Combien les piètres performances de ce compilateur coûtèrent-elles à National SemiConductor ?). Les compilateurs, les débogueurs et les éditeurs sont les trois principaux outils du programmeur dans sa vie quotidienne. gcc, gdb et Emacs étaient tellement supérieurs aux outils propriétaires que je ne pouvais m'empêcher de songer à la quantité d'argent qu'on pourrait gagner en substituant aux outils propriétaires ces programmes meilleurs et aux progrès plus rapides.

Voici une autre citation du Manifeste :

> Il n'y a rien de mal à souhaiter voir son travail rémunéré, ou à chercher à maximiser son revenu, tant que l'on n'utilise pas de moyens destructifs. Mais les moyens d'ordinaire utilisés pour les logiciels sont destructifs.
>
> Soutirer de l'argent aux utilisateurs d'un programme en en restreignant l'usage est destructif car les restrictions réduisent les usages

> potentiels du logiciel. Elles réduisent donc la quantité de richesse que l'humanité peut tirer du logiciel. Ces restrictions délibérées sont donc nuisibles et destructrices.
>
> La raison pour laquelle un citoyen ne doit pas employer de moyens destructifs pour s'enrichir est que si tout le monde faisait ainsi, nous serions tous appauvris par cette destruction mutuelle.

Bien qu'il ne soit pas écrit avec des pincettes, le "manifeste GNU" est en fait un document rationnel. Il dissèque la nature profonde des logiciels, de la programmation, de la grande tradition universitaire, et conclut à une obligation morale de partager gratuitement l'information qu'on a reçu gratuitement, indépendamment des conséquences financières. J'ai atteint une conclusion différente, qui m'a valu de longues discussions avec Stallman. Si je pense que la liberté d'utiliser de distribuer et de modifier le logiciel prendra le pas sur tout modèle visant à limiter la liberté, ce n'est pas pour des raisons éthiques mais à cause de la loi du marché.

Au début j'essayais d'argumenter de la même manière que Stallman : en vantant les qualités des logiciels libres. J'expliquais comment la liberté était le moteur d'une plus grande innovation à des coûts plus bas, permettait des économies d'échelle grâce à des standards plus ouverts, etc. Et les gens répondaient : « C'est une grande idée mais ça ne fonctionnera pas car personne ne voudra payer pour des logiciels gratuits. ». Après deux ans passés à perfectionner ma rhétorique, à affûter mes arguments, à porter la bonne parole aux gens qui me payaient pour voyager dans le monde entier, je n'avais jamais été plus loin que "C'est une grande idée, mais...", lorsque j'eus ma seconde illumination : si tout le monde pense que c'est une idée géniale, c'est sans doute une idée géniale, et si personne ne pense que ça va marcher, je n'aurai pas de concurrent !

> - F = - ma
>
> Isaac Newton

Vous ne verrez jamais un livre de physique où la loi de Newton est introduite de cette manière ; pourtant, mathématiquement parlant c'est strictement équivalent à « F = ma ». Ce que je veux dire par là, c'est qu'en faisant bien attention aux signes, on peut inverser les membres d'une équation sans remettre en cause sa validité. Faire de l'argent en offrant une assistance de nature commerciale pour des logiciels gratuits paraissait impossible parce que les gens, obnubilés par ce signe « - », oubliaient de le compter de l'autre côté du signe égal.

> On peut résister à l'invasion d'une armée, mais pas à celle d'une idée dont le temps est venu.
>
> Victor Hugo

Avant de laisser tomber ma thèse à Stanford pour créer une entreprise, il ne me restait plus qu'à répondre une question, largement hypothétique au demeurant. Supposons un instant que j'aie assez d'argent pour acheter n'importe quelle technique propriétaire (celle de Sun ou Digital, par exemple). Combien de temps pourrais-je gagner de l'argent avant que quelqu'un d'autre lance une entreprise avec les outils GNU et m'expulse du marché ? Pourrais-je seulement récupérer mon investissement initial ? Quand j'ai compris combien être en compétition avec des logiciels libres est une position inconfortable, j'ai su que le temps était venu.

> La différence entre la théorie et la pratique est très petite. . .En théorie.
>
> Anonyme

Dans cette section, je détaillerai la théorie du logiciel libre comme modèle commercial et la façon dont nous avons essayé de la mettre en pratique. Nous commencerons par quelques réflexions célèbres :

> Les marchés libres sont auto-organisés, ce qui permet un usage optimal des ressources et une création de valeur maximale
>
> Adam Smith

> L'information, quel que soit l'argent qu'on a dépensé pour la créer, peut être dupliquée et partagée avec un coût faible, voire nul.
>
> Thomas Jefferson

Le concept d'une économie du libre marché est si vaste que j'aime à plaisanter, chaque année, lors de la remise du Nobel d'économie, en disant qu'il est allé à celui qui a paraphrasé Adam Smith le plus élégamment. Mais toute blague mise à part, il existe un potentiel immense et inexploité, qui n'attend que d'être mis en valeur par un marché du logiciel vraiment libre.

Du temps d'Adam Smith, le libéralisme économique concernait surtout les particuliers : les marchés plus vastes, et spécialement les échanges entres nations, étaient très contrôlés. Lorsque les hommes d'affaires en eurent assez du système royal, ils se sont révoltés et ont créé un nouveau système de gouvernement qui s'occupait moins de leurs affaires. De fait, leur conception de la liberté dicta la Constitution des États-Unis, et elle semble partout à l'origine de nombre d'actions politiques et économiques. Qu'est-ce qui rend la liberté si attirante ? Et pourquoi liberté et prospérité économique sont-elles si étroitement liées ?

> Plus on comprend ce qui ne va pas dans une figure, plus cette figure est instructive.
>
> Lord Kelvin

Clairement, en ce qui concerne les outils pour programmeurs en 1989, les logiciels propriétaires étaient dans un état lamentable. D'abord, les outils étaient primitifs et offraient des fonctionnalités limitées. Ensuite ils avaient souvent des limitations arbitraires qui pouvaient devenir rédhibitoires lorsque les projets étaient complexes. Enfin, l'assistance technique proposée par les éditeurs de logiciels était horrible. À moins d'être un gros client et de pouvoir utiliser le pouvoir du portefeuille pour faire pression, on était sans recours quand on tombait sur un os. Enfin, chaque vendeur avait ses propres extensions propriétaires, ce qui faisait que quand on avait commencé à utiliser les maigres fonctionnalités d'une plate-forme, on se liait à elle, d'abord de façon imperceptible, puis de manière tout-à-fait inextricable. De fait, le modèle des logiciels propriétaires fonctionnait si mal que c'était intéressant à observer.

Encore de nos jours, l'économie de marché fonctionne à l'intérieur des murs des éditeurs de logiciels propriétaires (où les ingénieurs, les groupes de production sont en compétition pour obtenir des fonds et des avantages). À l'extérieur, l'utilisation et la distribution des logiciels est étroitement contrôlée par

des brevets, des licences, et des secrets commerciaux. On peut seulement imaginer l'énergie et l'efficacité qui sont perdues par cette pratique de la liberté au niveau micro-économique seulement.

> 1% d'inspiration, 99% de transpiration
>
> Thomas Edison

Dans la vision simpliste des éditeurs de logiciels, une fois que vous avez créé un logiciel assez bon pour que des gens veuillent l'acheter, faire des copies de ce logiciel est peu ou prou équivalent à copier des billets de banque : ça ne coûte quasiment rien et ça peut rapporter gros. Je pense qu'une des raisons pour lesquelles les logiciels étaient si misérables dans les années 1980 est que les gens se focalisaient sur le modèle idéal de l'impression de billets de banque, sans ce soucier de ce qui arriverait quand les gens commenceront à utiliser les liquidités. L'assistance était considéré comme un sous-produit dégénéré des défauts du logiciels, et minimiser son coût revenait donc à maximiser les bénéfices.

C'était frustrant pour les utilisateurs, mais tout aussi néfaste aux logiciels eux-mêmes. Des fonctionnalités pourtant faciles à ajouter avaient souvent été rejetées pour cause de "manque d'intérêt stratégique". Sans accès au code source, des fonctionnalités que les clients auraient été capables d'ajouter eux-mêmes restaient objets de spéculation et de contestation. Et au final les département marketing des éditeurs définissaient seuls l'espace de la concurrence, avec une myriade de fonctions inutiles mais tape-à-l'il. Le libéralisme économique marchait à l'envers !

> Personne n'a le monopole de la vérité.
>
> Anonyme

> La loi commune est un code légal qui est accessible à tous gratuitement.
>
> Michael Tiemann

C'est très bien d'avoir des théories merveilleuses pour bâtir un monde meilleur. C'est une autre paire de manches que d'amener ces théories au point où elles se tiennent debout d'elles-mêmes. Bien que les entreprises de service soient rares à cette époque, il y avait des exemples à étudier dans d'autres domaines.

Regardons la pratique de la justice aux États-Unis (ou en Grande-Bretagne). La loi commune est disponible gratuitement pour quiconque veut l'utiliser. Pas besoin de licence pour utiliser un texte de loi dans son argumentaire. De fait, les décisions législatives, quel qu'ait été le coût de leur production, sont accessibles à tous. Pourtant, les avocats sont parmi les professionnels les plus chers qui soient. Comment un avocat peut-il faire autant d'argent sans détenir aucun savoir propriétaire ?

Ce n'est pas seulement l'acte de poursuivre en justice que les gens estiment tant. C'est la valeur cumulée de ces actes. Si vous engagez un bon avocat et qu'une décision de justice est prise en votre avantage,cette décision devient un précédent qui fait jurisprudence et s'ajoute à la loi. La justice n'est pas aveugle, elle est chargée d'histoire.

Le travail des avocats n'est pas sans analogies avec la création et la maintenance de standards pour les logiciels libres. Créer un bon standard coûte très

cher. Mais travailler sans aucun standard ou maintenir un mauvais standard coûte bien davantage ! D'où l'intérêt d'avoir des gens valables pour travailler sur les logiciels qui seront les standards de demain. Au début, nous croyions que les gens comprendraient cet intérêt, et nous paieraient pour créer des logiciels libres de grande qualité qui deviendraient les standards de facto dans le monde du logiciel.

6.1 Les débuts de Cygnus

Ayant bien exploré la théorie, il était temps de la mettre en pratique. Créer une entreprise de services est assez facile, pour peu qu'on sache un peu gérer une entreprise. Malheureusement, aucun des trois fondateurs n'avait d'expérience dans ce domaine.

> Ne commettez jamais que des erreurs jusqu'alors inconnues.
>
> Esther Dyson

Nous avons lu des livres sur la création d'entreprise pour savoir enregistrer notre société, établir nos statuts, et autres formalités. Pour chaque penny sur lequel nous avons radiné durant la première année, nous avons dû débourser des milliers de dollars par la suite. Il n'est cependant pas évident que nous aurions pu faire mieux en achetant des conseils professionnels, car le premier de ces conseils que nous avons pris nous a coûté des centaines de dollars par heure, plus des dizaines de milliers pour se débarrasser définitivement du problème. Ces chiffres en disent long sur notre incapacité à juger l'importance des problèmes légaux et corporatifs et à demander les bons conseils, mais aussi sur la singulière incompétence des nombreux avocats avec lesquels nous avons parlé.

Ayant créé un modèle économique complètement nouveau, nous avons également créé nos concepts pour la finance, le marketing, les ventes, l'information des clients, et l'assistance. Chacune de ces créations nous a bien servi pendant la première année, si chaotique qu'elle fût, et chacun faisait ce qui était nécessaire pour faire décoller l'entreprise, mais ces concepts ont dû être complètement revus quand l'entreprise a grandi.

> Cygnus Nous rendons les logiciels gratuits moins chers
>
> John Gilmore

Pour combattre le chaos, nous avons travaillé dur à simplifier le principe de notre business : nous proposions un service d'assistance technique reconnu pour des logiciels techniques reconnus, en utilisant les économies d'échelle pour faire des bénéfices. D'après notre estimation, nous proposions une assistance deux à quatre fois meilleure que celle des ingénieurs maison, à un tarif deux à quatre fois moindre. Nous ne mettions pas trop l'accent sur les histoires de logiciels libres parce que c'était encore bien trop flou pour être vendable. Nous nous efforcions de donner aux gens de meilleurs outils pour moins cher, et contrat par contrat, nous avons appris comment faire.

Nous avons signé notre premier contrat en Février 1990, et à la fin du mois d'avril, nous avions pour 150000 dollars de contrats. En mai, nous avons envoyé des lettres à 50 candidats potentiels pour notre service et en juin, à 100 autres.

Soudain, l'entreprise devenait une réalité. À la fin de la première année nous avions pour 725,000 dollars de contrats et partout surgissaient de nouvelles occasions.

Ce franc succès a entraîné de sérieux problèmes. En vendant nos services pour la moitié ou le quart de ce que coûterait la même chose en interne, nous avions signé des contrats pour 1,5, voire 3 millions de dollars. Mais nous n'étions que 5 : un vendeur, un étudiant à temps partiel, et les trois fondateurs qui s'occupaient de tout, depuis le branchement des câbles Ethernet jusqu'à la conception des lettres à en-tête. En suivant le même taux de croissance, combien de nuits blanches nous faudrait-il pour en arriver là ? Personne ne le savait, car nous n'avions ni modèle financier, ni méthode opérationnelle.

6.2 GNUPro

Nous avons décidé de réaliser des économies d'échelle avant que l'usure devienne un problème. En vrais ingénieurs, nous avons décidé que le moyen le plus rapide de concrétiser ces économies était de se concentrer sur le plus petit sous-ensemble de logiciels libres que nous pouvions raisonnablement vendre comme une solution utile. Plus il serait petit, pensions-nous, et puis le déploiement en grandeur réelle serait facile.

> Avant tout, établis une base ferme.
>
> Sun Tzu

En faisant l'impasse sur les outils du shell, les utilitaires, les programmes de contrôle de version, et même sur un noyau libre pour le 386 d'Intel, nous avons décidé de vendre le compilateur et le débogueur GNU comme un produit prêt-à-l'emploi. Il y avait une bonne dizaine d'entreprises qui vendaient des compilateurs 32 bits fabriqués par d'autres, et autant de compilateurs conçus par des sociétés comme Sun, HP, ou IBM. En nous jetant dans la mêlée, nous avions l'impression qu'en dominant le marché des compilateurs 32 bits, nous deviendrions assez gros pour nous lancer dans d'autres projets (une gamme complète de logiciels libres, analogue au modèle de sous-traitance d'EDS pour les systèmes IBM).

Le compilateur GNU existait déjà sur des dizaines d'environnements et permettait de compiler vers plus de 10 architectures cibles (j'avais réalisé 6 de ces portages moi-même), ce qui en faisait un des compilateurs ayant le plus de portages au monde. Le débogueur GNU fonctionnait sur 5 plates-formes pour le code natif, et plusieurs personnes l'avaient adapté pour les systèmes embarqués. Fort naïvement, nous pensions que fabriquer un produit tout fait consisterait juste à rassembler des octets de divers provenances en une distribution, écrire un README et un script d'installation, ajouter quelques produits auxiliaires, tester le tout, et le distribuer. La réalité s'avéra beaucoup moins rose.

D'abord, gcc était dans la difficile transition entre la version 1.42 et la 2.0. La version 1 de gcc était meilleure que la plupart des compilateurs sur les machines CISC comme le 68000 ou le VAX. Mais pour la rendre compétitive sur les puces RISC, beaucoup d'optimisations étaient nécessaires. Le premier portage de gcc pour le processeur SPARC, en 1988, était 20 fois plus lent que le compilateur de Sun. En 1989 j'ai écrit un séquenceur d'instructions qui a ramené l'écart

à 10 et la même année, en optimisant les branchements, j'ai encore gagné 5. Avec la transition générale du CISC vers le RISC, ce qui était le meilleur compilateur sous tous les aspects devenait un choix moins évident, dont le client devait évaluer les avantages et les inconvénients. C'était beaucoup plus difficile à vendre.

Ensuite, GNU C++ était en perte de vitesse. J'ai écrit GNU C++ en 1987, c'était alors le premier compilateur de code natif pour C++. Le C++ est un langage beaucoup plus complexe que le C, et il était encore en mutation quand nous avons lancé Cygnus. En 1990, plusieurs fonctionnalités nouvelles, encore plus complexes, devinrent « standard » et avec le surmenage lié à Cygnus, je n'avais pas le temps de le maintenir à jour.

Troisièmement, gdb était en mille morceaux. Alors que gcc et g++ sont restés raisonnablement cohérents, avec des nouvelles versions émanant régulièrement d'un site central, gdb souffrait de fragmentation. Les adversaires du logiciel libre argumentent souvent en disant que l'avantage des logiciels propriétaires est qu'il n'y a qu'une version « officielle », tandis que chacun fait ses bitouilles dans son coin avec les logiciels libres, sans que se dégage un « standard » légitime. En l'absence d'un responsable, il se fragmentait donc, et des milliers de gens avaient leur propre version.

Quatrième difficulté, nous n'avions pas une chaîne d'outils complète. Nous avions un assembleur, un éditeur de liens, et d'autres outils (les fameux binutils) qui fonctionnaient sur certaines des plates-formes prises en charge par gcc et gdb, mais pas sur toutes, loin s'en fallait. En fait, si on faisait l'intersection des plates-formes prises en charge par gcc avec celles qui peuvent employer gdb, puis avec gas, gld, etc. on obtenait l'ensemble vide.

Cinq, nous n'avions pas de bibliothèque C, ce qui n'était pas un problème pour les plates-formes natives comme Sun ou HP, mais un enjeu capital pour les développeurs de systèmes embarqués, qui avaient besoin de construire des applications autonomes.

Sixième obstacle : nos concurrents ne pouvaient certainement pas égaler nos prouesses d'ingénierie just-in-time, mais ils avaient tous des produits déjà complets qu'ils vendaient avec beaucoup d'efficacité, chacun dans son marché de niche. En mettant sur pied et en vendant un produit tout fait, nous changions de plan : ce n'était plus une attaque sur le flanc, mais un assaut frontal contre des entreprises engendrant 10 ou 100 fois notre chiffre d'affaires.

Enfin, avions-nous confiance nous-mêmes en ce que nous faisions ? Ce qui est agréable quand on joue le rôle des intégrateurs de beaucoup d'outils évoluant très vite, c'est qu'on ressent très nettement que les gens ont besoin de vous. Les esprits sceptiques contestaient la notion même d'un logiciel libre tout emballé prêt-à-l'emploi en disant : dès que nous aurons sorti un produit qui franchira les tests de qualité, notre assistance deviendra inutile et en 6mois nous n'aurons plus de travail. C'était un défi pour notre métier que j'ai entendu maintes fois dans les quatre années qui ont suivi.

> Le Monde est plein de possibilités insurmontables.
>
> Yogi Berra

Nous n'avions rien de mieux à faire que de nous mettre au travail, et avec un calendrier initial de 6 mois, nous sommes tombés d'accord pour mettre les bouchées doubles afin d'y arriver. J'étais chargé de mener la barque de jour,

et la nuit je collaborais au travail pour gcc2.0 et g++. David Henkel-Wallace (alias Gumby), le second fondateur de Cygnus, s'occupait des binutils et de la bibliothèque en plus de ses occupations comme Directeur Général et Directeur du « Support ». Et John Gilmore, le troisième larron, a pris gdb. Nous avons engagé du monde pour nous aider à :

- 1. placer tout le travail sous CVS (CVS est un logiciel libre de contrôle de l'évolution du code source),
- 2. écrire des scripts d'installation et de configuration qui recouvriraient les centaines de plates-formes compatibles avec notre produit fini,
- 3. automatiser les procédures de test
- 4. nous aider à honorer les nouveaux contrats de développement dont la date limite se rapprochait sans cesse plus vite.

Six mois plus tard, nous avions incroyablement avancé dans notre travail, et certains commencèrent à trouver trop restrictive notre focalisation sur un unique produit (gcc). Le compilateur GNU resta notre principal fonds de commerce, mais nous avons commencé à signer des contrats pour d'autres logiciels, comme Kerberos (un système de sécurité réseau), Emacs, et même pour notre système de test et de recherche de bogues (qui était encore en cours de développement à l'époque).

John avait publia un article Usenet dont voici un résumé : « je suis le nouveau responsable de gdb. Si vous voulez que les fonctions que vous avez programmé fassent partie de la prochaine version et soient maintenues dans le futur, envoyez-moi les sources complètes et je verrai comment les intégrer ». En six semaines, il reçut 137 versions de gdb (des bidouillages de la version 3.5 pour la plupart), chacune ayant une ou plusieurs fonctionnalités ajoutées. Il a commencé à concevoir gdb4.0 afin d'inclure toutes ces fonctions supplémentaires. Comment aurais-je osé lui dire que c'était impossible, puisqu'il l'a fait ?

Gumby a décidé que tous les utilitaires pour travailler sur les formats binaires devraient utiliser la même bibliothèque, qui décrirait tous les fichiers objet et formats de débogage connus. La raison de ce choix est claire quand on regarde les fonctions des différents outils qui interviennent en aval du compilateur :

Utilitaires de manipulation de fichiers binaires

Outil	**Lit le format**	**Écrit le format**
gcc	ASCII	ASCII
as	ASCII	Binaire
ar	Binaire	Binaire
ld	Binaire	Binaire
size	Binaire	Binaire
strip	Binaire	Binaire
binary	Binaire	Binaire
Nm	Binaire	Binaire
gdb	Binaire	Binaire

Chacun de ces outils avait sa propre méthode pour lire ou écrire des fichiers au format binaire, et ils ne géraient pas tous les mêmes architectures et les mêmes formats : a.out, b.out, coff, ecoff, xcoff, elf, ieee695, et j'en passe. Si on faisait une correction dans l'assembleur pour puce 68000 au format a.out, on devait la réécrire dans tous les outils agissant sur ce format. Les changements étaient parfois faits de façon générique pour certains outils et de façon spécifique au format a.out pour d'autres !

En construisant une bibliothèque unique qui rassemblait toutes les fonctions dans le même code source, on pouvait réaliser plus vite des économies d'échelle car tous les outils auraient un comportement cohérent. De plus, ce serait possible de lier un fichier objet a.out avec une librairie au format coff pour produire un exécutable ieee695 ! Gumby a commencé à écrire cette bibliothèque. Il en a discuté avec Stallman, qui a affirmé que ce serait trop difficile il fallait réécrire tous les outils et la maintenance serait impossible. Gumby lui a répliqué que ce p. . .de job n'était pas infaisable (d'où le nom de la bibliothèque : BFD pour Big F**cking Deal, ou Binary File Descriptor, au choix).

Pendant que John et Gumby hackaient, j'avais toujours à vendre des contrats pour garantir une source de revenus. Chaque trimestre j'avais de nouveaux objectifs à atteindre et devais donc signer plus de contrats, et tous nos meilleurs ingénieurs étaient occupés par cette version pour laquelle nous n'avions aucune visibilité. Des tensions naquirent entre le secteur des ventes et celui des développeurs lorsque le modèle du logiciel libre paraissait fonctionner à l'envers : plus on s'occupait des logiciels GNU, moins on recevait de choses par l'Internet ; nous assurions plus de 50% du développement de la chaîne de logiciels GNU.

Ce n'était pas non plus une situation temporaire. Il aura fallu un an et demi avant d'arriver à la première version officielle. En ce jour mémorable, j'étais certain que pour la première fois, un atelier de développement C/C++ complet pouvait être construit à partir d'un code source unique sur deux plates-formes : Sun3 et Sun4. J'étais stupéfait : j'avais écrit six portages de gcc, 3 portages de gdb, un compilateur natif et un débogueur C++ en moins de temps qu'il en avait fallu à une équipe de hackers pour obtenir deux ensembles d'outils cohérents ! ?

Il y avait des circonstances atténuantes : premièrement, les outils marchaient mieux que jamais, et offraient plus de fonctionnalités ; deuxièmement, à cause du travail architectural accompli (nous avions non seulement réécrit les outils, mais aussi développé un script de configuration et mis en place toute une infrastructure de test automatique), il était théoriquement plus facile de réaliser des portages pour d'autres combinaisons hôte/cible, y compris pour un nombre illimité de systèmes embarqués.

Nous avons mis notre système à l'épreuve, et il a passé tous les tests haut la main :

Nombre de systèmes pris en charge au cours du temps

Date	**Nom de code**	**Systèmes natifs**	**Systèmes embarqués**	**Total**
Mars 1992	p1	2	0	2
Juin 1992	p2	5	0	5
Septembre 1992	p3	5	10	15
Décembre 1992	p4	5	20	25
Mars 1993	q1	5	30	35
Juin 1993	q2	5	45	50
Septembre 1993	q3	7	53	60
Décembre 1993	q4	8	67	75
Mars 1994	r1	10	75	85
Juin 1994	r2	10	80	90
Septembre 1994	r3	10	85	95
Décembre 1994	r4	10	90	100

Pendant que nos ingénieurs faisaient des prouesses pour créer GNUPro, notre équipe de commerciaux cherchait le meilleur moyen de le vendre. En 1991, nous avons engagé une jeune étudiante, récemment licenciée par Applied Materials, qui voulait apprendre comment vendre des logiciels. Bien que l'anglais ne soit pas sa langue maternelle, elle s'y est mis très vite. Étant tout sauf une bidouilleuse (bien qu'elle ait passé quelques week-ends à apprendre le C), elle est devenue l'un des meilleurs avocats des logiciels libres. Après six mois de succès, elle m'a invité à assister à une de ses présentations aux clients. J'étais estomaqué. J'avais toujours vendu les logiciels libres comme un hacker, en insistant sur les mérites techniques. Elle, de son côté, vantait la complexité du travail que nous faisions et la valeur des logiciels que nous fournissions, pour expliquer finalement aux clients qu'ils devraient acheter chez nous plutôt que d'essayer de travailler avec leurs ingénieurs. Je mettais en avant le fait que nos ingénieurs étaient meilleurs que les leurs (un message que peu de décideurs apprécient), alors qu'elle expliquait comment leurs ingénieurs pourraient bénéficier du travail de fond que nous accomplissions quand au portage, à la configuration et à la maintenance. Au total, en combinant les prouesses techniques et une bonne politique commerciale, Cygnus connut une grande croissance :

Croissance économique de la société

Réservations	**Rentabilité (%)**	**Taux de croissance**
1990 : 725	epsilon	N/A
1991 : 1500	1	106%
1992 : 2800	2	96%
1993 : 4800	3	87%
1994 : 5700	4	67%

Watson ! Viens ici !

Alexander Graham Bell

Issues de notre effort, des technologies importantes ont été rendues à l'Internet et sont devenues des standards en eux-mêmes : GNU configure (un script de configuration générique tenant compte de trois paramètres indépendants : le système sur lequel on compile le logiciel, le système hôte et la plate-forme cible), autoconf (un script de haut niveau pour créer des scripts configure), automake (un générateur de Makefile pour des environnements gérés par autoconf), Deja-GNU (une infrastructure pour les tests de non-régression), GNATS (un système de gestion des problèmes signalés), et j'en oublie.

Aujourd'hui, l'atelier GNUPro est compatible avec plus de 175 combinaisons hôte /cible, nombre qui plafonne à cause de la diversité actuelle du marché plutôt qu'à cause d'une limitation intrinsèque de notre technologie.

De fait, GNUPro est maintenant si bien installé que plusieurs de nos concurrents ont annoncé un service d'assistance pour les logiciels GNU, nous provoquant sur notre propre terrain ! Le modèle des logiciels libres vient heureusement une fois de plus à la rescousse. À moins qu'un concurrent puisse travailler autant que nos 100 ingénieurs, pour la plupart auteurs ou responsables de la maintenance des logiciels pour lesquels nous proposons du service, il ne peut nous chasser de la position de source GNU authentique (nous fournissons 80% des modifications apportées à gcc, gdb et consorts). Le mieux qu'il puissent espérer est d'ajouter des fonctions supplémentaires pour lesquelles leurs clients payeraient. Mais les logiciels étant des logiciels libres, toute modification ou valeur

ajoutée à ces logiciels revient gratuitement à Cygnus qui peut décider de l'intégrer ou non. Alors que le combat dans la sphère des logiciels propriétaires ressemble à un duel pour la vie ou la mort, la compétition sur des logiciels libres se déroule sur un ruban de Moebius : tout ce qui disparaît d'un côté réapparaît de l'autre, et finalement tout profite à l'auteur principal. Donc, si nos concurrents peuvent gagner certaines batailles grâce au "C'est un logiciel GNU, moi aussi, je peux en profiter", Cygnus reste le grand bénéficiaire sur le long terme. Ayant commencé en 1989, nous avons 10 ans d'avance sur tous les autres.

6.3 Les grands défis

Notre taux de croissance reste impressionnant, mais il s'est ralenti au fur et à mesure de notre développement. Pendant que nous essayions de vanter les vertus des logiciels libres, les sceptiques et les clients éventuels mettaient notre modèle à l'épreuve par rapport aux points suivants :

Bon sens

: Pourquoi un client payerait-il pour quelque chose qui avantagerait aussi ses concurrents ?

Croissance

: Comment une entreprise de services peut-elle gérer sa croissance ?

Pérennité

: Est-ce que Cygnus sera toujours là quand ses clients en auront besoin ?

Rentabilité

: Comment faire de l'argent avec des logiciels gratuits ?

Gestion

: Comment peut-on gérer des logiciels libres de manière à obtenir qualité et cohérence ?

Investissement

: Est-ce qu'une entreprise sans produit logiciel propre peut attirer les investisseurs ?

Essayez de vous imaginer la situation suivante : vous tentez de signer un contrat de 10000$ avec le responsable d'une équipe de 5 programmeurs pour systèmes embarqués, et il vous appelle pour savoir si oui ou non Cygnus peut émettre des actions dans le public étant donné son modèle commercial ! Autant, sur le plan technique, les logiciels libres nous ont ouvert des portes, étant les produits les plus aboutis et les plus innovants, autant ils se sont révélés un obstacle majeur lorsque nous avons attaqué le marché. Nous avons appris à nos dépens ce que Geoffroy Moore exprime dans son livre *Crossing the Chasm* (la traversée de l'abîme).

Cette difficulté m'est apparue clairement le jour où j'ai rendu visite à un groupe de programmeurs qui concevaient des réseaux de communication sans fil dans une entreprise qui figurait au Fortune 100. Dans le cadre de leur politique de qualité, ils évaluaient non seulement leur propre travail mais aussi leurs fournisseurs, selon un certain nombre de critères. La plupart de leurs fournisseurs étaient notés « Très bon » ou « Excellent » par rapport à tous les critères. Mais

le fournisseur des outils pour systèmes embarqués arrivait bon dernier avec un « Médiocre » ou « Inacceptable » dans chaque catégorie, et ce pour chacune des trois années où la qualité avait été évaluée. Pourtant, ils ne voulaient pas acheter nos outils, en dépit des recommandations (qui venaient de leurs propres clients !) que nous pouvions leur présenter, de notre supériorité technique et de nos prix imbattables. Pourquoi ? Parce que les responsables ne voulaient pas miser sur une solution déroutante, à laquelle ils n'étaient pas habitués. Je les ai laissés en leur demandant pourquoi ils se fatiguaient à évaluer la qualité s'ils n'utilisaient pas les données ainsi récoltées, mais ce n'était pas la bonne attitude. J'aurais plutôt dû réaliser que cette frilosité est un comportement typique du marché, et que la solution n'est pas de culpabiliser les clients, mais plutôt d'améliorer notre communication et notre marketing.

Nos problèmes ne venaient cependant pas seulement de l'extérieur. À tout moment de notre croissance, beaucoup de clients n'arrivaient pas à croire que nous arriverions à embaucher suffisamment de monde pour grandir bien au-delà de notre état actuel. Ils avaient à la fois raison et tort. Pour ce qui est des ingénieurs, ils avaient tort. Cygnus a été fondée par des ingénieurs, et notre culture d'entreprise, le pouvoir d'attraction des logiciels libres et la réputation de plus importante équipe à travailler sur les logiciels libres nous attiraient tous les développeurs que nous voulions engager. Et notre taux de renouvellement pour cause de départs était quelque chose entre le quart et le dixième de la moyenne nationale (surtout si l'on regarde du côté de la Silicon Valley).

Mais engager des commerciaux, c'était une autre paire de manches. La plupart partageaient les préjugés et les doutes de nos gros clients, et n'avaient aucune envie de travailler pour Cygnus. Ceux qui travaillaient pour nous le faisaient sans motivation. Ceux qui étaient motivés l'étaient souvent pour de mauvaises raisons. Il fut un temps où pour cinquante ingénieurs, nous n'avions que deux commerciaux. La communication, la gestion, et la satisfaction des employés déclinaient pendant que les commerciaux essayaient en vain de se familiariser avec ce que signifie la gestion d'une entreprise de logiciels libres.

Ironie du sort, nous avons aussi dû refuser la candidature de commerciaux qui n'acceptaient pas qu'on ajoute des composants propriétaires à nos produits. Les logiciels libres étaient une stratégie pour nous, pas une philosophie, et nous ne voulions pas engager de gens qui n'étaient pas assez souples pour gérer des produits libres ou non afin d'atteindre les grands objectifs de l'entreprise.

Nous avons fini par accepter le fait qu'on ne peut pas engager de commerciaux qui comprennent bien les tenants et les aboutissants du logiciel libre. Nous avons dû accepter leurs erreurs (ce qui signifie budgéter le coût de ces erreurs), et eux de tirer parti de ces erreurs pour progresser. La plupart d'entre eux arrivent avec une certaine expérience et tentent d'adapter la réalité afin qu'elle coïncide avec cette expérience source intarissable de problèmes pour Cygnus. Il était très difficile de trouver des gens capables de gérer d'après leur expérience mais aussi d'apprendre sur le tas, et assez vite. Et nous avions besoin de dizaines de commerciaux !

Le modèle des logiciels libres, ainsi mis à l'épreuve, s'est révélé à la fois très souple et très résistant. Bien que nous ayons parfois perdu des clients dont les attentes étaient déçues ou mal remplies, notre rendement annuel est resté approximativement à 90 par dollar depuis 1993, et la principale cause du départ des clients resta la cessation d'activité après la fin normale du contrat. Deux choses nous ont aidés à survivre là où d'autres n'auraient pas résisté : d'abord

chaque employé, indépendamment de son grade, reconnaissait à quel point la satisfaction des besoins des clients est importante ; personne n'était « au-dessus » de l'assistance à la clientèle. Ensuite, lorsque toute autre solution avait échoué, il restait au client la possibilité de se dépatouiller tout seul (puisque tous les clients avaient le code source). Ainsi, malgré d'innombrables difficultés durant ces années-là, très peu de clients ont été laissés complètement sans issue parce que le logiciel ne faisait pas son travail. Contraste énorme si on songe aux histoires que nous avons entendues à propos de logiciels propriétaires concurrents, ou encore de logiciels libres sans service d'assistance.

6.4 Trouver des fonds au-delà des logiciels libres — eCos

Dans le monde des systèmes embarqués, il y a un nombre relativement restreint de fondeurs (qui fabriquent les puces) et d'assembleurs (OEM). Le reste du marché se constitue de petits acteurs qui font des choses intéressantes, mais dont aucun n'a le volume nécessaire pour commander la conception de nouvelles puces ou de nouvelles solutions logicielles.

Entre les fondeurs et les assembleurs on trouvait des centaines de petits éditeurs de logiciels, chacun vendant son produit. Par exemple, il existe plus de 120 systèmes d'exploitation Temps Réel (SETR) commerciaux. Aucun ne dépasse les 6 de part de marché, selon l'IDC. C'est comparable au monde Unix il y a dix ans, en dix fois plus fragmenté. Cette situation conduit à une dégénérescence bien connue de l'économie de marché : redondance, incompatibilités, escroqueries sur les prix, etc. Ce que les fondeurs et les assembleurs voulaient, c'était des standards pour accélérer le TTM (Time to Money) ; mais les éditeurs de systèmes d'exploitation Temps Réel demandaient ou trop de temps, ou trop d'argent, ou encore les deux.

Nous étions l'étoile montante du marché des systèmes embarqués : nous grandissions deux fois plus vite que le leader de notre marché, en laissant nos quatre principaux concurrents avec une croissance à un chiffre. Cependant, nous n'étions pas de vrais leaders du marché, nous n'étions pas reconnus comme tels. En 1995, après de longues discussions avec nos principaux clients sur ce qui n'allait pas dans le monde des systèmes embarqués, nous avons commencé à comprendre que seuls les compilateurs de débogueurs GNUPro pourraient résoudre vraiment leurs problèmes. Ce dont les clients avaient besoin, c'est une couche d'abstraction entre le silicium et les programmes, qui leur permette de ne voir que la bibliothèque C standard ou encore une API POSIX temps réel. C'était une nouvelle occasion d'étendre notre offre de manière non triviale.

Nous avons taillé nos crayons et pris note des faits évidents : plus de 120 éditeurs et plus de 1000 SETR « maison », cela signifiait que du point de vue technique, personne n'avait su créer un SETR suffisamment souple pour convenir à tous les usages ; et du point de vue commercial, nous avons noté que les royalties pour l'usage de ces SETR étouffaient le marché, donc notre SETR devrait être sans royalties. Autrement dit, pour consolider le marché autour de nos produits, nous avions besoin de créer une technologie entièrement nouvelle, utilisable mondialement. Et nous devions la donner en retour. Les responsables ont traîné des pieds pendant un an avant de finalement se lancer dans cette idée.

Une fois que nous avions décidé notre stratégie, nos commerciaux continuaient à nous titiller avec la question : « Comment allons-nous faire de l'argent avec ça ? ». Alors même que notre domination avec GNUPro se renforçait, ce n'était pas évident pour eux de savoir comment nous pourrions réitérer le même succès avec un système d'exploitation pour systèmes embarqués.

Nous avons fait ce que toutes les entreprises font lorsqu'elles sont face à des contradictions : nous avons posé des axiomes. En supposant que nous saurions toujours faire de l'argent, nous avons pensé aux N autres problèmes qu'il fallait résoudre pour aider nos clients et devenir numéro 1. Nous devions :

1. développer cette nouvelle technique magique de configuration,
2. construire le reste du système afin que les gens aient quelque chose à configurer,
3. réaliser le tout avant que l'occasion de conquérir le marché disparaisse.

Développer des logiciels coûte cher, et développer un produit donné selon un calendrier fixé coûte bien plus cher encore.

Lorsque nous avons lancé Cygnus, nous avions tous supposé que les investisseurs ne comprendraient jamais ce que nous faisions, et qu'on n'aurait pas besoin d'eux avant cinq ans, voire plus. Fort heureusement, nous étions dans l'erreur sur les deux tableaux.

Le premier membre de la direction venu de l'extérieur, Philippe Courtot, n'a pas mis longtemps à me présenter à des investisseurs en 1992. J'étais très ouvert avec chacun d'eux au sujet de notre modèle, de notre technologie, de nos objectifs, et j'expliquais aussi que nous étions une entreprise à fonds propres, et que nous n'avions pas besoin de leur argent. De fait, nous arrivions à augmenter notre rentabilité d'un point chaque année tout en gardant un taux de croissance de 80%. Roger McNamee, un analyste reconnu de l'industrie des logiciels, l'a exprimé très bien lorsqu'il a dit : « Je suis à la fois enthousiasmé et surpris par votre modèle commercial. Je suis emballé par la façon dont ça marche, mais plus j'y pense, et plus je me demande pourquoi je n'ai pas eu l'idée avant ! »

C'était bien sûr gratifiant de penser que nous avions si bien réussi que nous n'avions pas besoin de financement extérieur, mais en réalité, en 1996, nous avions ouvert de telles perspectives derrière notre produit "GNUPro" que nous avions besoin de nouveaux objectifs et de nouveaux partenaires.

Nous avons trouvé deux investisseurs, Greylock Management et August Capital, qui comprenaient qui on était et ce qu'on faisait, et pouvaient avancer assez d'argent pour mettre notre plan à exécution. Ils ont investi 6,25 millions de dollars, le plus gros placement privé dans une entreprise de logiciels de la première moitié de 1997, et la réalisation a suivi de près.

> I do not like them Sam-I-am. I do not like green eggs and ham.
>
> Dr.Seuss

Pendant que l'équipe technique travaillait, les commerciaux continuaient à bosser sur la manière de faire de l'argent, car au tout début nous n'avions pas vu le rapport entre l'architecture de l'eCos et le modèle commercial utilisé pour le vendre. Du point de vue technique, nous savions que la configurabilité était la clé de l'existence d'un produit « taille unique » qui conviendrait à tous. Du point de vue du commerce, nous savions qu'un système d'exploitation « taille unique » était le moyen d'unifier le marché autour d'un standard valable pour le développement des systèmes embarqués. Pendant un an et demi, chacun traitait

le problème de son côté. Incapable d'assumer le paradoxe des logiciels libres, beaucoup de commerciaux n'y arrivaient pas.

Lorsque les ingénieurs furent en mesure d'exhiber ce qu'ils avait seulement imaginé au début, c'est devenu clair pour tout le monde : nous étions en train de créer la première architecture ouverte, la première architecture libre. J'étais aussi excité que la première fois que j'ai vu gcc.

Les logiciels libres sont pain béni pour le bitouilleur et la manière dont ils créent des standards profite à l'utilisateur final, mais il n'y en a pas moins un gouffre entre ce que les hackers et ce que les simples utilisateurs peuvent faire avec des logiciels libres. Nous voulions qu'eCos soit adopté par les bitouilleurs, mais aussi par le programmeur moyen. Notre idée était de donner aux utilisateurs des outils de haut niveau pour configurer, personnaliser et valider eCos de façon automatique, remplaçant les étapes manuelles que les développeurs de SETR font aujourd'hui. En réalisant des outils qui contrôlaient eCos au niveau du code source, et en architecturant le code source de manière à ce qu'il soit administrable par ces outils, nous avons permis à l'utilisateur moyen de travailler au niveau du code source, sans même écrire une ligne de C ou C++. La preuve de notre succès est qu'eCos peut être configuré de manière à tenir dans 700 octets (strict minimum nécessaire pour réaliser une couche d'abstraction au-dessus du silicium) ou grandir jusqu'à 50 Ko (SETR complet avec IP et un système de fichiers !)

Une fois que nous avons compris que l'ouverture du code source n'était pas seulement une option, mais une caractéristique essentielle d'eCos, et qu'elle nous donnait un facteur 10 par rapport à la concurrence (10 fois moins de place en mémoire par rapport à la configuration au niveau objet, et 10 à 100 fois plus d'efficacité pour les programmeurs par rapport à des SE dont les sources sont disponibles, mais pas au cur de l'architecture), nous avons conçu différents emballages pour mettre notre produit sur le marché, et les premières réactions ont été extrêmement positives.

Si l'on considère les obstacles que nous avons du vaincre pour mettre en place notre activité autour des logiciels GNU, on ne peut qu'imaginer les potentialités de notre système eCos, pour Cygnus et pour le monde.

6.5 Réflexions et Perspectives

Les logiciels libres profitent de l'efficacité inhérente à un marché techniquement libre, mais d'une manière imprévisible, organique. Les entreprises peuvent jouer dans ce domaine le rôle de la "main invisible" d'Adam Smith, dirigeant l'évolution de manière à faire avancer le marché global tout en atteignant leurs objectifs au niveau micro-économique. Les réussites les plus spectaculaires dans ce domaine seront destinées à ceux qui pourront se lancer dans les technologies qui engendrent la plus grande coopération de la communauté de l'Internet et qui résolvent les plus grand défis techniques et économiques des utilisateurs.

Issu des logiciels libres, l'Internet a été un déclencheur fantastique pour le développement de nouveaux logiciels libres. À mesure que les gens continueront à se connecter à l'Internet et à utiliser les logiciels libres, nous allons assister à des changements dans le monde du logiciel comparables à ce que fut la Renaissance pour le monde des savants et des universitaires. Avec le vent de liberté qu'ils font souffler sur le monde, je n'en attends pas moins !

Il a travaillé à des arts inconnus, et a changé les lois de la nature.

James Joyce

Chapitre 7

Génie Logiciel

Le « génie logiciel » couvre un champ plus vaste que « l'écriture de programmes ». Néanmoins, dans de nombreux projets de logiciels libres, les programmes sont tout simplement écrits et diffusés. Il est clair, à partir d'exemples passés, que la réalisation d'un logiciel n'a pas besoin d'être organisée pour qu'il soit utilisé et apprécié. Dans cet essai, nous examinerons quelques éléments généraux du génie logiciel, puis leurs équivalents habituels dans la communauté des logiciels libres, pour finalement voir les implications des différences entre ces deux approches.

7.1 Le cycle de développement du logiciel

Les étapes suivantes permettent de décrire, en général, le cycle de développement du logiciel :

- L'expression des besoins du produit,
- La conception préliminaire, au niveau système,
- La conception détaillée,
- L'implémentation, ou phase de codage,
- L'intégration,
- Les essais in situ,
- La maintenance et assistance.

Aucune ne doit commencer avant que les précédentes ne soient réellement terminées, et lorsqu'une modification est effectuée sur un élément, tous ceux qui en dépendent doivent être revus en en tenant compte. Il se peut qu'un module donné soit à la fois spécifié et implémenté avant que ceux qui en dépendent soient complètement spécifiés, situation que l'on appelle développement avancé ou recherche.

Il est absolument essentiel que chaque phase du processus liée au génie logiciel subisse plusieurs types de revues : revue des pairs, revue par le responsable projet et par la direction, et revue interdisciplinaire.

Les éléments correspondant au cycle de développement du logiciel (les documentations ou code source) doivent porter des numéros de version et faire l'objet d'un historique. « L'enregistrement » d'une modification dans un élément impose un certain type d'examen dont l'ampleur doit correspondre directement à l'importance des modifications.

7.1.1 Expression des besoins

La première étape du cycle de développement d'un logiciel consiste à produire un document qui décrit les utilisateurs visés et leurs objectifs. Ce document formalise la liste des fonctions à accomplir pour répondre aux besoins des clients. Le « Dossier de Spécification du Logiciel » (DSL) constitue le document de référence dans lequel on trouvera les réponses aux questions « Que doit-on faire et qui utilisera le produit ? ».

Le DSL de nombreux projets qui échouèrent avait été considéré comme constituant les « Tables de la loi » remises par les gens du commercial aux ingénieurs, qui, alors, ronchonnèrent longuement sur les lois de la physique et sur les raisons qu'ils avaient de ne pas pouvoir fabriquer ce produit puisqu'ils n'était pas possible de s'approvisionner en « Kryptonite » ou quoi que ce soit d'autre. Le DSL est le fruit d'un effort conjoint auquel les ingénieurs doivent

aussi participer en rédigeant de nombreuses sections, et non en se contentant d'en analyser le termes.

7.1.2 Conception préliminaire

C'est une description de haut niveau du produit, en termes de « modules » (ou quelquefois de « programmes ») et de leurs interactions. Ce document doit en premier lieu asseoir la confiance en la finalité et la faisabilité du produit, et, en second lieu, servir de base pour l'estimation de la quantité de travail à fournir pour le réaliser.

Le « Dossier de Conception Préliminaire » doit également mettre en évidence le plan de test, en termes de besoins de l'utilisateur, et montrer que l'on peut y satisfaire grâce à l'architecture proposée.

7.1.3 Conception détaillée

C'est au niveau de la conception détaillée que chacun des modules énumérés dans le dossier de conception préliminaire est décrit en détail. L'interface (formats de lignes de commande, appels d'API, structures de données visibles de l'extérieur) de chacun des modules doit être complètement définie à ce niveau. Deux choses doivent émerger lors de cette étape : un diagramme de PERT ou de GANTT, montrant comment le travail doit être fait et dans quel ordre, ainsi qu'une estimation plus précise de la charge de travail induite par la réalisation de chacun des modules.

Chaque module doit avoir un plan de test unitaire, qui fournit aux réalisateurs la liste des tests à effectuer ou des types de scénarios de test a créer afin de vérifier que le module répond aux spécifications. Il faut noter qu'il existe des tests unitaires complémentaires, ne concernant pas les fonctionnalités, dont on parlera plus tard.

7.1.4 Implémentation

Chacun des modules décrit dans le document de spécification détaillé doit être réalisé. Cela comprend la petite activité de codage ou de programmation qui constitue le cur et l'âme du processus de développement du logiciel. Il est malheureux que cette petite activité soit quelquefois l'unique partie du génie logiciel qui soit enseignée (ou étudiée), puisque c'est également la seule partie du génie logiciel qu'un autodidacte peut réellement appréhender.

On peut considérer qu'un module a été réalisé quand il a été créé, testé et utilisé avec succès par un autre module (ou par un processus de test au niveau système). La création d'un module se fait dans le cadre du cycle classique édition-compilation-répétition. Le test des modules comprend les tests au niveau unitaire les tests de non-régression définis lors de la conception détaillée, ainsi que les tests de performances et de charge et l'analyse de couverture du code.

7.1.5 Intégration

Quand tous les modules sont terminés, l'intégration, au niveau du système, peut être réalisée. C'est là que tous les modules sont réunis en un seul ensemble

de code source, compilés et liés pour former un paquetage qui constitue le système. L'intégration peut être réalisée de façon incrémentale, en parallèle avec la réalisation de différents modules, mais on ne peut pas décider de manière autoritaire que « c'est fini » tant que tous les modules ne sont pas effectivement terminés.

L'intégration comprend le développement de tests au niveau du système. Si le paquetage réalisé est capable de s'installer lui-même (ce qui signifie simplement le décompactage d'une archive ou la copie de fichiers d'un CD-ROM) il doit alors exister un moyen de le faire automatiquement, soit sur des systèmes spécialisés, soit dans des environnements de simulation.

Parfois, dans le cas des logiciels personnalisés, le paquetage est constitué du simple exécutable résultant de la compilation de l'ensemble des modules, et, dans ce cas, il n'y a pas d'outil d'installation ; les tests seront effectués tels quels.

Le système ayant été installé (s'il doit l'être), le processus de tests au niveau système doit pouvoir lancer toutes les commandes publiques et appeler tous les points d'entrée publics, en utilisant toutes les combinaisons raisonnables d'arguments. Si le système doit pouvoir créer une sorte de base de données, la procédure automatisée de test au niveau système doit en créer une et utiliser des outils extérieurs (écrits séparément) pour vérifier l'intégrité de la base de données. Les tests unitaires peuvent être utilisés pour répondre à certains de ces besoins, et tous les tests unitaires doivent être exécutés en séquence pendant le processus d'intégration, de construction et de réalisation du paquetage.

7.1.6 Essais in situ

Les essais in situ commencent généralement en interne. Ce qui signifie que des employés de l'organisation qui a produit le logiciel vont l'essayer sur leurs propres ordinateurs. Ceci doit inclure tous les systèmes « du niveau production », c'est-à-dire tous les ordinateurs de bureau, les portables et les serveurs. Vous devez pouvoir dire, au moment où vous demanderez à vos clients d'utiliser le nouveau système logiciel (ou une nouvelle version d'un logiciel existant) « nous l'avons nous-même testé ». Les développeurs du logiciel doivent être disponibles lors de l'assistance technique directe assurée pendant la phase de tests in situ interne.

Enfin, il sera nécessaire de faire fonctionner le logiciel à l'extérieur, c'est-à-dire sur les ordinateurs des clients (ou de ceux que l'on espère voir devenir des clients). Il vaut mieux choisir des « clients amicaux » pour ce genre d'exercice, puisqu'ils découvriront peut-être de nombreux défauts, même évidents, tout simplement parce que leur manière de travailler et leurs habitudes sont différentes de celles de vos utilisateurs internes. Les développeurs du logiciel doivent être en première ligne pendant cette phase de test in situ externe.

Les défauts rencontrés pendant la phase de test in situ devront être étudiés par des développeurs confirmés et des ingénieurs commerciaux, pour déterminer ceux qui doivent être corrigés dans la documentation, ceux qui doivent être corrigés avant que cette version du logiciel ne soit diffusée et ceux qui le seront dans la prochaine version (ou jamais).

7.1.7 Maintenance et assistance

Les défauts du logiciel rencontrés soit pendant la phase de test in situ soit après sa diffusion doivent être enregistrés dans un système de suivi. Il faudra affecter un ingénieur logiciel pour la prise en charge de ces défauts, qui proposera de modifier soit la documentation du système, soit la définition d'un module ou la réalisation de ce module. Ces modifications devront entraîner l'ajout de tests unitaires ou au niveau système, sous forme de tests de non-régression pour mettre en évidence le défaut et montrer qu'il a bien été corrigé (et pour éviter de le voir réapparaître plus tard).

Exactement comme le DSL a constitué une entreprise en commun entre les commerciaux et les ingénieurs, la maintenance est une entreprise commune aux ingénieurs et au service client. La liste des bogues, la description de bogues particuliers, le nombre maximum de défauts critiques dans une version diffusée du logiciel,. . .constituent les pièces maîtresses de cette édifice.

7.2 Tests détaillés

7.2.1 Tests de couverture

Le test de couverture du code commence avec l'introduction d'instructions spéciales dans le code du programme, quelquefois par un préprocesseur, quelquefois par un modificateur de code objet, quelquefois en utilisant un mode spécial du compilateur ou de l'éditeur de liens, pour suivre tous les chemins possible dans un bloc de code source et d'enregistrer, pendant leur exécution, tous ceux qui ont été parcourus.

Examinons l'extrait de code C, tout à fait typique, suivant :

```
1.   if (read(s, buf, sizeof buf) == -1)
2.   error++;
3.   else
4.   error = 0;
```

Ce code présente un défaut si la variable « error » n'a pas été initialisée, et si la ligne 2 est exécutée les résultats seront imprévisibles. La probabilité d'apparition d'une erreur dans une instruction « read » (et d'obtenir une valeur de retour égale à -1) lors de tests normaux est assez faible. La manière d'éviter la maintenance coûteuse de ce genre de bogue consiste à s'assurer que les tests unitaires traitent tous les chemins possibles dans le code et que les résultats sont corrects dans tous les cas.

Mais il y a mieux : les chemins possibles dans le code se combinent. Dans notre exemple ci-dessus, la variable d'erreur peut avoir été initialisée avant, disons par un morceau de code similaire dont le prédicat (« échec de l'appel système ») était faux (signifiant l'apparition de l'erreur). L'exemple suivant, dont le code est manifestement incorrect et qui n'aurait pas surmonté un examen, montre la facilité avec laquelle des choses simples peuvent devenir compliquées :

```
1.   if (connect(s, sa, sa_len) == -1)
2.   error++;
3.   else
4.   error = 0;
5.   if (read(s, buf, sizeof buf) == -1)
6.   error++;
7.   else
8.   error = 0;
```

Il existe maintenant quatre chemins à tester dans le code :
- lignes 1-2-5-6
- lignes 1-2-5-8
- lignes 1-4-5-6
- lignes 1-4-5-8

Il est en général impossible de tester tous les chemins possibles dans le code, il peut y en avoir des centaines, même dans une petite fonction de quelques dizaines de lignes. Et d'un autre côté, il n'est pas suffisant de s'assurer uniquement que les tests unitaires sont capables (éventuellement en plusieurs fois) de mettre à l'épreuve toutes les lignes de code. Ce type d'analyse de couverture ne fait pas partie de la trousse à outils de tous les ingénieurs logiciel sur le terrain, c'est pourquoi l'assurance qualité (AQ) en fait sa spécialité.

7.2.2 Tests de non-régression

Corriger une erreur n'est pas suffisant. L'expression « évident à l'examen » constitue souvent une réponse destinée à cacher celle, plus insidieuse, affirmant « il serait difficile d'écrire un test qui pète le feu ». Bon, oui, de nombreux bogues sautent aux yeux lors d'un examen, comme la division par la constante zéro. Mais pour savoir quoi corriger, il faut examiner le code environnant pour comprendre les intentions de l'auteur. Ce genre d'analyse doit être documenté dans la correction ou dans les commentaires accompagnant le code source.

Dans le cas le plus fréquent, le bogue n'est pas évident à l'examen et la correction se trouve à un endroit différent, dans le code source, de celui où le programme provoque une erreur fatale ou a un comportement inadéquat. Dans ces circonstances, il faut écrire un nouveau test qui met à l'épreuve la partie de code défaillant (ou qui met le programme dans un état incorrect ou autre) puis il faut tester la correction avec ce nouveau test unitaire. Après revue et enregistrement, le nouveau test unitaire doit également être enregistré de façon à ce que si le même bogue est réintroduit plus tard comme effet secondaire de quelque autre modification, l'AQ ait quelque espoir de le trouver avant les clients.

7.3 Le développement des logiciels libres

Un projet de logiciel libre peut inclure n'importe lequel des éléments décrits ci-dessus, et pour être honnête, c'est le cas de quelques uns d'entre eux. Les versions commerciales de BSD, Bind et Sendmail sont des exemples d'un processus de développement logiciel standard, mais ils n'ont pas débuté de cette façon. Un véritable processus de développement logiciel nécessite énormément de ressources, et en lancer un représente, en général, un investissement que seul un retour peut justifier.

Le cas, de loin le plus fréquent, de développement d'un logiciel libre est celui où les gens concernés se régalent dans ce qu'ils font et veulent que leur travail soit aussi utile que possible. Et, pour ce faire, ils le diffusent sans rétribution et quelquefois sans restrictions sur sa distribution. Ces gars-là peuvent ne pas avoir accès à ce que l'on appelle des outils de « qualité commerciale » (tels que des analyseurs de code, des vérificateurs de limites et des contrôleurs de l'intégrité mémoire). Ils paraissent trouver la programmation, la réalisation de paquetage, et le prosélytisme particulièrement amusants, et pas l'AQ (Assurance Qualité), les DSL (Dossier de Spécification du Logiciel) et ils ne sont pas habituellement des champions des délais de réalisation courts.

Reprenons chacune des étapes du processus de développement d'un logiciel et voyons comment elles s'articulent dans un projet de logiciel libre, non financé, d'un travail de passionné.

7.3.1 Expression des besoins

Les développeurs de logiciel libre ont tendance à réaliser les outils dont ils ont besoin ou dont ils ont envie. Il arrive quelquefois que cela coïncide avec ce qu'ils font au travail et souvent ce sont ceux dont la tâche première est l'administration plutôt que le développement de logiciel. Si, après plusieurs versions, un système logiciel atteint la masse critique et devient autonome, il sera distribué sous forme de fichier archive via l'Internet et les utilisateurs pourront commencer, soit à demander l'ajout de fonctionnalités, soit à se mettre à en ajouter et à en faire profiter la communauté.

La constitution d'un « DSLL » (Dossier de Spécification d'un logiciel libre) s'effectue généralement grâce à une liste de diffusion ou à un groupe de discussion où utilisateurs et développeurs discutent directement. Le consensus se fait sur ce que les développeurs ont retenu de la discussion ou sur ce avec quoi ils sont d'accord. Si un consensus suffisant ne peut être obtenu, il va en résulter un « embranchement dans le code » où d'autres développeurs se mettront à diffuser leurs propres versions. L'équivalent du DSL dans le cas du logiciel libre peut être très enrichissant mais peut créer des conflits aigus dont la résolution n'est pas possible (ou pas tentée).

7.3.2 Conception préliminaire, au niveau système

Il n'y a pas, en général, d'étape de conception préliminaire dans un projet de développement d'un logiciel libre non financé. Cette étape est soit implicite, jaillissant toute entière et d'un seul coup de la cuisse de Jupiter, soit elle évolue avec le temps (comme le logiciel lui-même). Habituellement, avec les versions 2

ou 3 d'un logiciel libre, on voit apparaître une réelle architecture du système, même si celle-ci n'a pas été formalisée quelque part.

C'est ici, plutôt que n'importe où ailleurs que, par rapport aux règles normales du développement des logiciels, les logiciels libres ont gagné leur réputation d'être quelque peu farfelus. On peut compenser une absence de DSL ou même de démarche d'assurance qualité en ayant de très bons programmeurs (ou des utilisateurs particulièrement amicaux), mais, s'il n'y a pas eu de conception au niveau du système (même si c'est seulement dans la tête de quelqu'un), la qualité du projet résultant sera limitée.

7.3.3 Conception détaillée

Le DCD (Dossier de Conception Détaillée) constitue une autre victime de l'absence de rétribution et de la volonté de se faire plaisir des gens qui développent des logiciels libres. Certains aiment travailler sur un DCD mais, en général, prennent tout leur plaisir lorsqu'ils le font dans leur travail journalier. La conception détaillée devient un effet de bord du développement : « Je sais que j'ai besoin d'un analyseur, donc, je vais en écrire un ». La documentation des interfaces de programmation des applications (API) sous la forme de symboles externes, dans des fichiers d'en-tête ou dans des pages de man est optionnelle et peut devenir inexistante si cette interface n'est pas destinée à être publiée ou utilisée en dehors du projet.

C'est vraiment regrettable car cela rend inaccessible pas mal de très bons codes qui, autrement, pourraient être réutilisables. Même les modules qui ne sont pas réutilisables ou étroitement liés au projet pour lequel ils ont été créés, et dont les interfaces de programmation ne font pas partie des éléments à délivrer, devraient vraiment avoir des pages de man pour expliquer ce qu'ils font et comment les appeler. C'est extrêmement utile pour ceux qui veulent améliorer le code, puisqu'ils doivent commencer par le lire et le comprendre.

7.3.4 Implémentation

C'est la partie amusante. L'implémentation est la partie que les programmeurs préfèrent ; c'est ce qui les fait veiller alors qu'ils devraient dormir. L'occasion d'écrire du code constitue la motivation première de pratiquement tous les efforts de développement de logiciel libre qui ont eu lieu. Si l'on se concentre sur cet aspect du développement logiciel à l'exclusion des autres, il apporte une très grande liberté d'expression.

C'est dans le développement de logiciels libres que la plupart des programmeurs expérimentent de nouveaux styles, que ce soit un style d'indentation, une manière de donner un nom aux variables ou « un essai d'économie d'utilisation de la mémoire » ou « un essai d'économie de cycles CPU » ou ce que vous voulez. Et il y a quelques extraits d'une grande beauté qui attendent dans des archives un peu partout, où un programmeur a essayé avec succès un nouveau style.

Un effort pour un logiciel libre non financé peut avoir autant de rigueur et de qualité qu'on le veut. Les utilisateurs exploitent le code s'il est fonctionnel et la plupart des gens ne font pas attention au fait que le développeur a changé de style trois fois pendant la phase de réalisation. Les développeurs y font en général attention, ou apprennent à le faire après un certain temps. Devant une

telle situation, les commentaires de Larry Wall, selon qui la programmation est un art, prennent toute leur signification.

La principale différence dans une réalisation de logiciel libre non financé vient de ce que la revue du code est informelle. Il n'y a généralement pas de supérieur ou de pair pour regarder le code avant qu'il ne soit diffusé. Il n'y a habituellement pas de test unitaire, de non-régression ou quoi que ce soit d'autre.

7.3.5 Intégration

L'intégration, dans un projet logiciel libre, consiste habituellement à écrire quelques pages de man, à s'assurer que le logiciel se construit sur tous les systèmes auxquels le développeur a accès, à nettoyer le fichier « Makefile » pour le débarrasser de toutes les commandes superflues qui se sont accumulées pendant la phase d'implémentation, à écrire un LISEZMOI, à faire une archive, à la télécharger sur un site ftp anonyme quelque part et à poster une annonce dans la liste de diffusion ou le forum de discussion des utilisateurs concernés.

Notons que le groupe de discussion comp.sources.unix a été ranimé en 1998 par Rob Braun et qu'il constitue l'endroit idéal pour poster des annonces concernant de nouveaux paquetages de logiciels libres ou des mises à jour de logiciels existants. Il fonctionne également comme archive de dépôt.

Il n'y a pas, habituellement, de plan de test au niveau du système et pas de tests unitaires. En fait, pour ce qui est des logiciels libres, les efforts consentis pour les tests en général sont très limités (des exceptions existent, telles que Perl et PostgreSQL). Cependant, comme nous allons le voir, ce manque des tests avant diffusion n'est pas une faiblesse.

7.3.6 Tests in situ

Le logiciel libre non financé bénéficie des meilleurs tests au niveau système, sauf si nous y incluons les tests de la NASA sur les robots destinés à aller dans l'espace. Cela vient simplement du fait que les utilisateurs ont tendance à être beaucoup plus amicaux lorsqu'ils n'ont rien à payer, et les utilisateurs faisant autorité (souvent eux-mêmes développeurs) sont souvent plus serviables lorsqu'ils peuvent lire et corriger le code source de quelque chose qu'ils emploient.

Le manque de rigueur constitue l'essence des tests de terrain. Le génie logiciel impose aux utilisateurs qui réalisent les tests d'essayer des cas d'utilisation intrinsèquement imprévisibles au moment où le système a été conçu et réalisé ; en d'autres termes, des expériences réelles d'utilisateurs réels. Les projets de logiciels libres non financés sont proprement imbattables dans ce domaine.

La « revue des pairs » de dizaines ou de centaines d'autres programmeurs traquant les bogues en lisant le code source plutôt qu'en utilisant simplement des paquetages exécutables constitue un autre avantage dont profitent les projets de logiciels libres. Quelques uns des lecteurs rechercheront les failles de sécurité et certaines de celles qui seront découvertes ne seront pas signalées (en dehors du cercle des pirates), mais ce risque est faible par rapport à l'avantage de voir un nombre incalculable d'étrangers lire le code source. Ces étrangers peuvent réellement faire travailler le développeur plus tard le soir qu'aucun patron ou aucun chef ne pourrait ou ne voudrait l'imposer.

7.3.7 Maintenance et assistance

« Oh là là, désolé », voilà ce que l'on répond habituellement à un utilisateur qui découvre un bogue, ou, « Oh là là, désolé, et merci ! » si celui-ci ajoute la correction du bogue. Les développeurs de logiciels libres font le tri des bogues à la réception des messages « Hé, ça marche chez moi ». Cela vous paraît chaotique, ça l'est. Le manque d'assistance peut empêcher certains utilisateurs de vouloir (ou de pouvoir) utiliser les programmes issus du monde des logiciels libres non financés, mais cela peut également créer des occasions pour des cabinets de conseil ou des distributeurs de logiciels de vendre des contrats d'assistance et/ou de vente de versions améliorées et/ou de vente de versions commerciales.

Lorsque la communauté des éditeurs d'Unix s'est trouvée confrontée à la demande pressante, de la part de ses utilisateurs, de fournir des logiciels libres pré-intégrés dans leurs systèmes de base, leur première réaction a été de dire « Eh bien, d'accord, mais nous n'en assurerons pas l'assistance ». Le succès d'entreprises comme Cygnus ? a été dû à son ré-examen rapide de cette position, mais le fossé entre les cultures est vraiment très profond. Les éditeurs de logiciels traditionnels, y compris les éditeurs d'Unix, ne peuvent tout simplement pas planifier la fourniture de contrats d'assistance si de nombreuses personnes inconnues modifient librement les logiciels.

Quelquefois, la réponse consiste à reprendre le logiciel en interne, en le faisant entrer dans le cycle normal de l'assurance qualité avec les tests unitaires et les tests au niveau du système, l'analyse de couverture du code et tout le reste. Cela peut entraîner une ingénierie inverse du DSL et du DCD pour fournir une référence à l'assurance qualité (i.e., quelles fonctionnalités doivent être testées). D'autres fois, la réponse consiste à réécrire les termes du contrat d'assistance en remplaçant « résultats garantis » par « meilleurs efforts ». Finalement, le marché de l'assistance sera occupé par ceux qui pourront fédérer tous cette foule d'étrangers, dans la mesure ou parmi eux il y a des gens bien, nombreux, qui écrivent du logiciel de qualité, et que la « culture logiciel libre » est, dans de nombreux cas, la plus efficace pour fournir le niveau de fonctionnalité que les utilisateurs désirent réellement (comme en témoigne Linux par rapport à MS-Windows).

7.4 Conclusions

L'ingénierie est un domaine ancien et, que l'on réalise un logiciel, un matériel ou un pont de chemin de fer, les étapes du processus sont essentiellement les mêmes :

- Identification d'un besoin, et des demandeurs,
- Conception d'une solution qui réponde à ce besoin,
- Division du projet en modules, planification de la réalisation,
- Réalisation, tests, fourniture et suivi.

Certains domaines privilégient certaines phases. Par exemple, les constructeurs de ponts de chemin de fer n'ont pas trop à réfléchir sur le document récapitulant les besoins du marché (le DSL, pour le logiciel), le processus de réalisation, ou le suivi, mais il doivent faire très attention au dossier de spécification détaillé, au dossier de conception détaillé et, bien évidemment à l'assurance qualité (AQ).

Le moment déterminant de la conversion d'un « programmeur » en « ingénieur logiciel » se situe lorsqu'il réalise que le génie logiciel est un domaine dans lequel il peut entrer mais que, pour cela, il lui faut une approche intellectuelle radicalement différente, et pas mal de travail. Il se passe souvent des années avant que la différence entre programmation et génie logiciel ne saute aux yeux des développeurs de logiciels libres, simplement parce qu'il faut plus de temps pour que les projets de logiciels libres souffrent du manque de rigueur dans leur développement.

Ce chapitre décrit le génie logiciel de façon superficielle afin de le présenter au développeur de logiciel libre, et de susciter son intérêt. Le futur, ne l'oublions pas, est toujours constitué du meilleur du passé et du présent. Le génie logiciel n'est pas une sorte de règle à calcul ou de calculatrice de poche mais un mode de réalisation de systèmes de haute qualité, riche de techniques nombreuses et éprouvées, particulièrement efficace dans le cas de projets non réalisables dans le cadre de l'approche par un « unique programmeur de génie » commune à de nombreux développements de logiciels libres.

Chapitre 8

Linux à la pointe du progrès

Linux, aujourd'hui, compte des millions d'utilisateurs, des milliers de développeurs, et sa part de marché augmente. Il est utilisé dans des systèmes embarqués et pour contrôler des robots, il a voyagé à bord de la navette spatiale. Je voudrais pouvoir dire que je l'avais prévu, que cela s'inscrivait dans un projet de domination du monde, mais à vrai dire j'ai été un peu surpris. Le passage du nombre d'utilisateurs de Linux de un à cent m'est apparu beaucoup plus nettement que la transition entre ces derniers et le million suivant.

La réussite de Linux ne dépend pas d'une disposition préalable visant à le rendre portable et répandu mais plutôt de l'efficacité des principes gouvernant sa conception et son mode de développement. Ces fondations solides ont permis d'obtenir plus facilement portabilité et disponibilité.

Comparons Linux aux projets commercialement soutenus, par exemple Java ou MS-Windows NT. L'engouement créé par Java a convaincu beaucoup de personnes que write once, run anywhere (écrivez une fois, exécutez partout) vaut quelques sacrifices. Nous entrons dans une période où un nombre croissant de matériels informatiques différents seront utilisés, il s'agit donc bien d'un concept important. Sun n'a cependant pas inventé cette notion de "write once, run anywhere". La portabilité est depuis longtemps le saint Graal de l'industrie informatique. Microsoft, par exemple, a d'abord espéré que MS-Windows NT devienne un système d'exploitation portable pouvant fonctionner sur les machines Intel mais aussi sur les processeurs RISC, courants dans les stations de travail. Linux n'a jamais eu un objectif si ambitieux. C'est donc une ironie du sort que Linux soit devenu une plate-forme efficace à ce titre.

Linux a tout d'abord été conçu pour une seule architecture : l'Intel 386. Il fonctionne aujourd'hui sur tout type de machine, du PalmPilot à la station Alpha ; il s'agit du système d'exploitation porté sur le plus grand nombre d'architectures disponible sur PC. Si vous écrivez un programme pour Linux, il pourra être « write once, run anywhere » pour un grand nombre de machines différentes. Il est intéressant d'observer les décisions prises lors de la conception de Linux et la manière dont son développement a évolué afin de voir comment il a réussi à se transformer en quelque chose qui n'était pas du tout prévu à l'origine.

8.1 Le portage sur Amiga et Motorola

Linux est un système d'exploitation de type Unix, mais il ne s'agit pas d'une version d'Unix, ce qui lui confère un héritage différent de, par exemple, celui de FreeBSD. Les créateurs de ce dernier ont débuté avec le code source de Berkeley Unix et leur noyau en est directement issu. Il s'agit donc d'une version d'Unix, dans la lignée directe de ses prédécesseurs. Linux, en revanche, a pour but de fournir une interface compatible avec Unix, mais le noyau a été complètement réécrit, sans aucune référence au code source d'Unix. Ainsi, Linux n'est pas un port d'Unix, mais un nouveau système d'exploitation.

Je n'avais vraiment pas l'intention, au début, de porter ce nouveau système d'exploitation sur d'autres plates-formes. En premier lieu, je voulais obtenir quelque chose qui fonctionne sur mon 386.

Le premier effort de taille fait pour rendre le code du noyau de Linux portable a eu lieu afin de commencer son port sur les machines Alpha de DEC. Il ne s'agissait toutefois pas du premier port.

Le premier port a été réalisé par une équipe qui rendit disponible le noyau Linux sur la famille Motorola 68000 (puce présente dans les premiers ordinateurs Sun, Apple et Amiga). Les programmeurs responsables voulaient vraiment faire quelque chose de bas niveau et en Europe, beaucoup de personnes dans la communauté Amiga étaient particulièrement rétives à MS-DOS ou MS-Windows.

Même si la communauté Amiga a réussi à obtenir un système fonctionnant sur les 68000, je ne considère pas vraiment qu'il s'agisse d'un port réussi de Linux. Ils ont pris le même style d'approche que celle que j'avais adoptée lorsque j'ai commencé à écrire Linux ; écrire, sans aucune référence, un code destiné à assurer un certain type d'interface. Ainsi, ce premier port pour 68000 pouvait être considéré comme un système d'exploitation ressemblant à Linux, un embranchement du code d'origine.

De ce point de vue, ce premier port 68000 de Linux n'a pas aidé à créer un Linux portable, mais il y contribua d'une autre manière. Quand j'ai commencé à penser au port pour Alpha, j'ai dû étudier l'expérience du port 68000. Si nous prenions la même approche avec les Alpha, j'aurais alors à maintenir trois codes de base différents. Même si cela aurait été possible au niveau de la programmation, ce n'était pas gérable. Je ne pouvais veiller au développement de Linux s'il fallait pour cela maintenir une nouvelle base de code chaque fois que quelqu'un voulait Linux sur une nouvelle architecture. À la place, je voulais faire un système où j'aurais une branche spécifique pour les Alpha, une autre pour les 68000 et une troisième pour les x86, toutes articulées autour d'une base de code commune.

C'est pourquoi il y eut une réécriture importante du noyau à cette époque, qui fut aussi motivée par la volonté de permettre à un nombre croissant de développeurs de travailler de concert.

8.2 Micro-noyaux

Quand j'ai commencé à écrire le noyau de Linux une théorie arrêtée définissait la manière d'écrire un système portable. Tout le monde préconisait d'utiliser une architecture fondée sur les micro-noyaux.

Dans les noyaux monolithiques, par exemple celui de Linux, la mémoire est divisée entre l'espace utilisateur et l'espace noyau. L'espace noyau est l'endroit où le code réel du noyau réside après son chargement, et où la mémoire est allouée pour les opérations qui prennent place à son niveau. Ces opérations incluent l'ordonnancement, la gestion des processus et des signaux, des entrées/sorties assurées par les périphériques, de la mémoire et de la pagination. Ce sont autant d'opérations de base requises par les autres programmes. Le code du noyau assure les interactions de bas niveau avec le matériel, les noyaux monolithiques semblent spécifiques à une architecture donnée.

Un micro-noyau effectue un nombre bien plus restreint d'opérations, et sous une forme plus limitée : la communication entre processus, une gestion limitée des processus et de l'ordonnancement ainsi qu'une partie des entrées/sorties de bas niveau. Ainsi, les micro-noyaux se révèlent être moins spécifiques au matériel car un grand nombre des particularités du système sont placées dans l'espace utilisateur. Une architecture à micro-noyau est en quelque sorte une manière de s'éloigner des détails du contrôle des processus, de l'allocation mémoire et de celle des ressources, afin que le port vers un autre matériel ne requière que des

changements minimes.

Ainsi, au moment où j'ai commencé à travailler sur Linux, en 1991, on supposait que la portabilité devait découler d'une approche de type micro-noyau. Elle constituait alors la marotte des chercheurs en informatique théorique. Cependant, je suis pragmatique et pensais alors que les micro-noyaux

1. étaient expérimentaux,
2. étaient manifestement plus complexes que les noyaux monolithiques
3. avaient une vitesse d'exécution bien inférieure à celle des noyaux monolitiques.

La rapidité importe beaucoup dans la réalité, c'est pourquoi on dépensait beaucoup d'argent dans le domaine de la recherche sur les optimisations pour micro-noyaux afin qu'ils puissent fonctionner aussi rapidement que leurs pendants plus classiques. Une des choses amusantes à la lecture de ces articles est qu'on s'apercevait que bien que les chercheurs appliquent leurs astuces d'optimisation sur des micro-noyaux, on pouvait aussi facilement utiliser ces dernières sur des noyaux traditionnels pour les accélérer.

En fait, cela m'a amené à estimer que l'approche fondée sur des micro-noyaux était fondamentalement malhonnête car destinée à obtenir davantage de subsides pour la recherche. Je ne pense pas nécessairement que ces chercheurs étaient consciemment malhonnêtes. Peut-être étaient-ils tout simplement stupides, ou induits en erreur. Je le dis dans son sens premier : la malhonnêteté vient de la forte pression qui s'exerçait à ce moment précis à l'intérieur de la communauté scientifique pour favoriser l'étude des micro-noyaux. Dans un laboratoire d'informatique, vous étudiiez les micro-noyaux ou bien vous n'étudiiez pas du tout les noyaux. Ainsi, tout le monde, y compris les concepteurs de MS-Windows NT, était embarqué dans cette approche malhonnête. Alors même que cette équipe savait que le résultat final ne ressemblerait en rien à un micro-noyau, elle était consciente qu'elle devait lui rendre des hommages peu sincères.

Heureusement, je n'ai jamais subi de réelle pression pour aller vers les micro-noyaux. Les chercheurs de l'université d'Helsinki étudiaient les systèmes d'exploitation depuis la fin des années 1960 et ne trouvaient plus grand intérêt dans la recherche sur les noyaux. Dans un sens, ils avaient raison : les bases des systèmes d'exploitation, et par extension du noyau Linux, étaient bien comprises au début des années 1970 ; tout ce qui a été fait ensuite n'a été, d'une certaine manière, qu'un exercice d'auto-satisfaction.

Si vous voulez rendre un code portable, vous n'avez pas nécessairement besoin de créer un niveau d'abstraction pour assurer la portabilité. Vous pouvez aussi programmer intelligemment. Essayer de rendre les micro-noyaux portables est par essence une perte de temps, cela revient à construire une voiture exceptionnellement rapide équipée de pneus carrés. L'idée de s'abstraire de la partie qui doit être ultra rapidele noyau est de façon inhérente contre-productive.

Bien entendu, dans la recherche sur les micro-noyaux, il y a plus que cela. Mais la partie cruciale du problème est la différence d'objectifs. Le but de la majeure partie de la recherche sur les micro-noyaux était de concevoir un idéal théorique, de définir un mode de conception aussi portable que possible sur toutes les architectures imaginables. Je ne caressais pas, avec Linux, d'objectif aussi noble. J'étais intéressé par la portabilité entre des systèmes réels et non théoriques.

8.3 De l'Alpha à la portabilité

Le portage Alpha a débuté en 1993 et dura environ un an. Il n'était pas complètement achevé au bout d'un an, mais toutes les bases s'y trouvaient. Alors que ce premier port était difficile, il a établi quelques principes de conception gouvernant Linux depuis, qui ont rendu les autres portages plus simples.

Le noyau Linux n'est pas écrit pour être portable sur toute architecture. J'ai décidé que si une architecture était suffisamment saine et satisfaisait certaines règles simples, alors Linux devrait pouvoir y être adapté sans problème majeur. Par exemple, la gestion de la mémoire peut être très différente d'une machine à l'autre. J'ai lu des documents décrivant la gestion de la mémoire des 68000, Sparc, Alpha et PowerPC et trouvé, malgré des différences dans les détails, qu'il y avait beaucoup de points communs dans l'utilisation de la pagination, du cache etc. Il suffisait que la gestion de la mémoire du noyau Linux soit écrite sur la base du dénominateur commun de toutes ces architectures pour qu'il ne soit ensuite pas trop difficile de modifier ce code pour régler les détails propres à une architecture particulière.

Quelques suppositions simplifient beaucoup le problème des ports. Si, par exemple, vous déclarez qu'un microprocesseur doit pouvoir utiliser la pagination il doit disposer d'une sorte de table de traduction (en anglais Translation Lookup Buffer ou TLB), qui lui dit à quoi correspond la mémoire virtuelle. Bien entendu, vous ne pouvez savoir quelle sera la forme de la TLB. Mais en fait, les seules choses que vous devez connaître sont les méthodes pour la remplir et pour la purger lorsque vous décidez que vous n'en avez plus besoin. Ainsi, dans cette architecture bien pensée, vous savez que vous aurez besoin de placer dans le noyau quelques parties spécifiques à la machine, mais que le plus gros du code exploite des mécanismes généraux sur lesquels reposent des choses comme la TLB.

Une autre règle que j'applique est qu'il est toujours préférable d'utiliser une constante de compilation plutôt qu'une variable et, en utilisant cette règle, le compilateur fait souvent une bien meilleure optimisation du code. Il est évident que c'est plus sage, car vous pouvez écrire votre code afin qu'il soit défini de façon facile à adapter et à optimiser.

Ce qu'il y a d'intéressant dans cette approcheessayer de définir une architecture commune bien pensée est que, ce faisant, vous pouvez présenter au système d'exploitation une architecture meilleure que celle qu'offre le matériel. Cela semble aller à l'encontre de toute intuition, mais c'est important. Les généralisations que vous cherchez en résumant les systèmes sont souvent les mêmes que les optimisations que vous voudriez faire pour améliorer les performances du noyau.

En fait, quand vous faites un résumé suffisamment important de problèmes comme la mise en uvre des tables de pagination et quand vous prenez une décision fondée sur vos observationspar exemple, le fait que l'arbre de pagination doit être seulement de profondeur trois, vous vous apercevez plus tard que c'était la seule approche raisonnable pour obtenir de bonnes performances. En d'autres termes, si vous ne vous étiez pas intéressé à la portabilité en tant que critère de conception, mais aviez simplement essayé d'optimiser le noyau sur une architecture donnée, vous auriez souvent atteint la même conclusion c'est-à-dire que la profondeur optimale pour représenter l'arbre de pagination au niveau du noyau

est trois.

Cela ne procède pas seulement du hasard. Quand une architecture diffère par certains détails d'une conception générale fonctionnelle, c'est souvent parce qu'elle est mal conçue. Ainsi, les mêmes raisons qui vous font passer outre les particularités de conception à des fins de portabilité vous obligent aussi à contourner les défauts de conception tout en préservant une conception générale plus optimisée. J'ai essayé d'atteindre le juste milieu en combinant le meilleur de la théorie avec la réalité des architectures actuelles.

8.4 Espace noyau et espace utilisateur

Avec un noyau monolithique comme celui de Linux, il est important de faire très attention lors de l'introduction de nouveau code et de nouvelles fonctionnalités. Ces décisions peuvent ensuite affecter de nombreuses choses dans le cycle de développement, au delà du noyau lui-même.

La première règle simple est d'éviter les interfaces. Si quelqu'un veut ajouter quelque chose qui implique une nouvelle interface système, vous devez être exceptionnellement attentif. Dès lors que vous fournissez une interface aux utilisateurs, ils vont commencer à l'utiliser et lorsque quelqu'un aura commencé à coder par dessus cette interface, vous ne pourrez plus vous en débarrasser. Voulez-vous maintenir exactement la même interface pour le reste de la vie de votre système ?

Le reste du code n'est pas si problématique. S'il n'a pas d'interface, par exemple un pilote de disque, alors il n'y a pas besoin d'y réfléchir longtemps ; vous pouvez sans grand risque l'ajouter simplement. Si Linux n'avait pas ce pilote auparavant, l'ajouter ne gênera pas les utilisateurs antérieurs, et cela l'ouvre à de nouveaux utilisateurs.

Quand la question se pose sur d'autres plans, vous devez faire des compromis. Est-ce une bonne implémentation ? Est-ce que cela ajoute vraiment une intéressante fonctionnalité ? Parfois, même si la fonctionnalité est bonne, il se trouve que soit l'interface est mauvaise, soit l'implémentation est telle que vous ne pourrez jamais faire autre chose, maintenant ou plus tard.

Par exemplebien que ce soit en quelque sorte une question d'interface supposez que quelqu'un a une implémentation stupide d'un système de fichiers où les noms ne peuvent pas avoir plus de 14 caractères. Ce que vous voulez vraiment éviter est d'avoir, pour une interface, ces limitations gravées dans le marbre. Sinon, lorsque vous voudrez étendre le système de fichiers, vous serez foutu car devrez trouver une manière de rester dans cette petite interface jusqu'alors figée. Pire que cela, tout programme qui demande un nom de fichier peut avoir réservé (par exemple) 13 caractères pour une variable et ainsi, si vous passez un nom de fichier plus long, vous le planterez.

À présent, le seul éditeur qui fasse une chose aussi stupide est Microsoft. En simplifiant, pour lire des fichiers MS-DOS/MS-Windows, vous avez cette ridicule interface où les noms de fichiers comptent onze caractères, huit plus trois. Avec NT, qui a permis les noms longs, ils ont dû ajouter un ensemble complet de nouvelles routines pour faire la même chose que les anciennes, sauf que cet ensemble pouvait aussi prendre en compte de plus grands noms de fichiers. Cela constitue par là-même un exemple d'interface incorrecte car gênante lors des travaux ultérieurs.

Un autre exemple de cette situation est donné dans le système d'exploitation Plan 9. Ils avaient cet appel système vraiment pratique pour faire une meilleure duplication des processus. En deux mots, ce nouveau fork, appelé par Plan 9 R-Fork (et que SGI a appelé plus tard S-Proc) crée deux espaces d'exécution distincts partageant le même espace d'adressage. Cela facilite en particulier le threading.

Linux permet aussi cela avec son appel système « clone », mais il a été implémenté de façon correcte. Les concepteurs des routines SGI et Plan9 ont décidé que les programmes ayant deux branchements pouvaient partager le même espace d'adressage, mais séparèrent leurs piles. Normalement, lorsque vous utilisez la même adresse dans les deux threads, vous obtenez la même position mémoire. Mais le segment de pile est spécifique de sorte que si vous utilisez une adresse mémoire de la pile, vous obtenez en fait deux positions mémoires différentes, qui peuvent partager un pointeur de pile sans écraser l'autre pile.

Ce comportement est astucieux, mais a son revers : le surcoût de la maintenance des piles le rend, en pratique, vraiment inadéquat. Ils se sont aussi aperçus trop tard que les performances s'avèrent catastrophiques. Comme des programmes utilisaient cette interface, il n'y avait plus moyen de la modifier. Il leur a fallu introduire une interface supplémentaire, correctement écrite, de façon à utiliser l'espace de pile d'une façon raisonnable.

Un éditeur commercial peut de temps en temps essayer de faire passer le défaut de conception dans l'architecture mais Linux n'offre pas ce recours.

La gestion du développement de Linux et la prise de décisions de conception imposent la même approche. D'un point de vue pratique, je ne pouvais gérer le fait que de nombreuses personnes fournissent des interfaces au noyau. Je n'aurais pas été capable d'en garder le contrôle. Mais du point de vue conception, c'était aussi ce qu'il fallait faire : garder le noyau relativement petit et maintenir au minimum le nombre des interfaces et des autres contraintes imposées au développement futur.

Bien entendu, Linux n'est pas complètement « propre » de ce point de vue. Il a hérité des versions précédentes d'Unix un certain nombre d'interfaces affreuses. Ainsi, dans certains cas, j'aurais été bien plus heureux de ne pas avoir à maintenir la même interface qu'Unix. Mais Linux est à peu près aussi propre que peut l'être un système sans repartir complètement de zéro. Et si vous voulez profiter de la possibilité de faire fonctionner des applications Unix, il faut bien en accepter certaines lourdeurs. La possibilité de faire fonctionner ces applications a joué un rôle primordial dans la popularité de Linux, le jeu en valait donc la chandelle.

8.5 GCC

Unix lui-même est une belle histoire à succès en ce qui concerne la portabilité. Le noyau Unix, comme de nombreux noyaux, repose sur l'existence du C pour lui donner la plus grande partie de sa portabilité, à l'instar de Linux. La disponibilité de compilateurs C sur de nombreuses architectures a permis d'y porter Unix.

Unix montre donc bien l'importance des compilateurs. C'est l'une des raisons pour lesquelles j'ai choisi de placer Linux sous GPL (GNU Public License). Elle protège aussi le compilateur GCC. Je pense que tous les autres programmes du projet GNU sont, du point de vue de Linux, insignifiants en comparaison. GCC

est le seul dont je me préoccupe vraiment et j'en déteste un certain nombre ; l'éditeur Emacs, par exemple, est horrible. Bien que Linux occupe plus d'espace qu'Emacs, il a au moins dans son cas l'excuse que c'est nécessaire.

Mais les compilateurs sont vraiment un besoin fondamental.

Maintenant que la structure générale du noyau Linux facilite son portage, tout au moins vers des architectures raisonnablement conçues, il devrait être possible de l'adapter à une quelconque plate-forme, pour autant que l'on dispose d'un bon compilateur. Je ne m'inquiète plus d'éventuels problèmes structurels qui pourraient entraver l'adaptation du noyau aux processeurs à venir mais plutôt des compilateurs. Le processeur 64 bits d'Intel appelé Merced en est un exemple frappant, car il est très différent des processeurs actuels du point de vue d'un compilateur.

Ainsi, la portabilité de Linux est très liée au fait que GCC est porté sur les architectures les plus répandues.

8.6 Modules noyau

Il est très vite devenu évident que nous voulions développer un système aussi modulaire que possible. Le modèle de développement à Sources Libres l'impose, parce que sinon, vous ne pouvez pas facilement laisser plusieurs personnes travailler simultanément. Il est vraiment pénible d'avoir plusieurs personnes qui se gênent les unes les autres parce qu'elles travaillent sur la même partie du noyau.

Sans modularité je devrais vérifier tous les fichiers modifiés afin de m'assurer que ces changements n'affectent pas autre chose. Cela représenterait un travail énorme. Avec la modularité, quand quelqu'un m'envoie des corrections pour un nouveau système de fichiers et que je ne lui accorde pas confiance a priori, je peux toujours être certain que si personne n'utilise ce système de fichiers, il n'aura aucun impact sur d'autres parties.

Ainsi, Hans Reiser développe un nouveau système de fichiers, et vient d'arriver à le faire fonctionner. Je ne pense pas qu'il vaille la peine d'être inséré dans le noyau 2.2 dès maintenant. Mais, grâce à la modularité du noyau, je le pourrais si je le voulais vraiment et ce ne serait pas trop difficile. Il s'agit surtout d'empêcher les gens de se marcher sur les pieds.

Avec le noyau 2.0, Linux a vraiment pris du poids. C'est le moment où nous avons ajouté les modules dans le noyau. Ceci a de façon évidente amélioré la modularité en donnant une structure explicite pour écrire des modules. Les programmeurs pouvaient travailler sur différents modules sans risque d'interférence. Je pouvais garder le contrôle de ce qui était écrit dans le noyau à proprement parler. Ainsi, encore une fois, la gestion des intervenants et celle du code ont amené à prendre la même décision de conception. Pour que les personnes travaillant sur Linux restent coordonnées, nous avions besoin d'une solution ressemblant aux modules noyau. Mais du point de vue de la conception, c'était aussi la chose à faire.

L'autre aspect de la modularité est moins évident, et plus problématique. Il s'agit du chargement dynamique, que tout le monde apprécie mais qui pose de nouveaux problèmes. Le premier problème est d'ordre technique, et est donc, comme d'ordinaire, (presque) toujours le plus simple à résoudre. Le problème le plus important concerne les parties non techniques. Par exemple, à quel point un module est-il un travail dérivé de Linux et par conséquent sous GPL ?

Quand la première interface des modules a été faite, des personnes ayant écrit des pilotes pour SCO ne voulaient pas en publier le code source, comme requis par la GPL. Néanmoins, ils étaient prêts à le recompiler pour fournir des binaires pour Linux. À ce point, pour des raisons morales, j'ai décidé que je ne pouvais pas appliquer la GPL à ce genre de situation.

La GPL impose que les travaux « dérivés » d'un travail sous GPL soient protégés par elle. Malheureusement, la définition d'un travail dérivé est assez vague. Dès que vous essayez de tracer une frontière pour les travaux dérivés, le problème se transpose immédiatement au choix de l'endroit où placer la frontière.

Nous avons fini par décider (à moins que ce soit moi qui aie fini par décréter) que les appels systèmes ne seraient pas considérés comme un lien au noyau. Tout programme fonctionnant au-dessus de Linux ne pouvait donc pas être considéré comme couvert par la GPL. Cette décision a très rapidement été prise et j'ai même ajouté un fichier spécial LISEZMOI (voir Annexe B) pour en informer tout le monde. Pour cette raison, les éditeurs commerciaux peuvent écrire des programmes pour Linux sans avoir à s'inquiéter de la GPL.

Le résultat, du point de vue des programmeurs de modules, était qu'on pouvait écrire un module propriétaire si l'on n'utilisait que l'interface normale pour le chargement. Il s'agit cependant toujours d'une zone d'ombre du noyau. Ces zones d'ombre laissent peut-être toujours des occasions aux gens pour abuser de la situation et c'est en partie à cause de la GPL qui n'est pas particulièrement claire à propos de thèmes tels que l'interface de modules. Si quelqu'un devait passer outre en utilisant les symboles exportés dans le seul but de contourner la GPL, alors je pense qu'il y aurait une possibilité de poursuites en justice. Mais je ne pense pas que quiconque veuille utiliser le noyau de façon malhonnête ; ceux qui ont montré un intérêt commercial dans le noyau l'ont fait parce qu'ils étaient intéressés par les avantages apportés par le modèle de développement.

La puissance de Linux réside autant dans la communauté qui coopère afin de l'améliorer que dans le code lui-même. Si Linux était détourné si quelqu'un voulait faire et distribuer une version propriétaire son attrait, qui est principalement le modèle de développement à Sources Libres, serait perdu dans cette version propriétaire.

8.7 La portabilité aujourd'hui

Aujourd'hui, Linux a atteint nombre des buts de conception dont certains supposaient à l'origine qu'ils ne pourraient être atteints que sur une architecture micro-noyau.

En construisant un modèle général de noyau fondé sur des éléments communs à des architectures répandues, le noyau Linux a obtenu beaucoup des avantages de la portabilité qui auraient autrement demandé l'ajout d'un niveau d'abstraction, et ce sans souffrir de la même perte de performances que les micro-noyaux.

En autorisant les modules noyau, le code spécifique au matériel peut souvent être confiné, en laissant ainsi le code du noyau hautement portable. Les pilotes de périphériques sont un bon exemple de l'usage effectif de modules noyau pour confiner les spécificités matérielles aux modules. C'est un bon compromis entre mettre toutes les spécificités matérielles dans le noyau de base (ce qui le rend rapide mais non portable) et les mettre dans l'espace utilisateur (ce qui donne

un système lent ou instable, voire les deux).

Mais l'approche de la portabilité adoptée par Linux a été aussi bénéfique à tout son environnement de développement. Les décisions qui ont motivé la portabilité ont aussi permis à un large groupe de personnes de travailler simultanément sur des parties de Linux sans pour autant que j'en perde le contrôle. Les généralisations d'architectures sur lesquelles se fonde Linux m'ont donné une base de référence pour y confronter les changements du noyau, et fournissent suffisamment d'abstraction pour que je n'aie pas à maintenir des parties complètement séparées du code pour chaque architecture. Donc, même si beaucoup de personnes travaillent sur Linux, je peux gérer le noyau en lui-même. Et les modules noyau fournissent aux programmeurs une solution évidente pour travailler indépendamment sur des parties du système qui se doivent d'être indépendantes.

8.8 L'avenir de Linux

Je suis sûr que nous avons pris la bonne décision avec Linux en lui faisant effectuer le moins d'opérations possibles en espace noyau. La vérité est que je n'envisage pas de changements majeurs dans le noyau. Un projet logiciel réussi doit arriver à maturité à un certain point, et alors le rythme des changements ralentit. Il n'y a plus beaucoup d'innovations majeures en stock pour le noyau. Il s'agit plutôt d'offrir un éventail plus large pour le choix du système matériel : profiter de la portabilité de Linux pour le rendre disponible sur de nouveaux systèmes.

De nouvelles interfaces naîtront, mais je pense qu'elles proviendront en partie du port vers un plus grand nombre de systèmes. Par exemple, quand vous commencez à faire des grappes (clusters), vous voyez apparaître le besoin de coordonner l'ordre d'exécution de certains ensembles de processus, ce qui impose une coopération entre les ordonnanceurs. Mais en même temps, je ne veux pas que tout le monde se focalise sur les grappes et les super-ordinateurs parce que les ordinateurs portables, les cartes que vous branchez où que vous alliez et d'autres développements de ce genre peuvent jouer un grand rôle dans le futur et je voudrais donc que Linux aille aussi dans cette direction.

Les systèmes embarqués, eux, ne proposent pas la moindre interface utilisateur. Vous n'accédez au système que pour changer le noyau, et encore, mais le reste du temps, vous n'y touchez pas. Il s'agit là encore d'une autre direction pour Linux. Je ne pense pas que Java ou Inferno (le système d'exploitation embarqué de Lucent) investiront le secteur des périphériques embarqués. Ils n'ont pas saisi l'importance de la loi de Moore . En premier lieu, concevoir un système optimisé spécifique à un périphérique embarqué particulier peut sembler judicieux, mais dans le même temps, la loi de Moore dit qu'un matériel plus performant sera disponible pour le même prix, détruisant ainsi la plus-value apportée par la conception pour un périphérique particulier. Tout devient si bon marché que vous pourriez aussi bien avoir le même système sur votre ordinateur de bureau que sur votre périphérique embarqué. Cela rendra la vie plus facile à tout le monde.

Les systèmes multi-processeurs symétriques représentent un des domaines qui va être développé. Le noyau Linux 2.2 va gérer très correctement quatre processeurs et nous l'améliorerons jusqu'à pouvoir utiliser huit ou seize processeurs. La gestion de plus de quatre processeurs est déjà présente, mais pas

tout-à-fait achevée. Si vous disposez de plus de quatre processeurs maintenant, vous jetez votre argent par la fenêtre. Donc ce sera certainement amélioré.

Mais si les gens veulent soixante-quatre processeurs ils devront utiliser une version spécifique du noyau, parce que les gérer dans le noyau normal causerait une perte de performances pour l'utilisateur ordinaire.

Certains domaines d'application particuliers continueront à être le fer de lance du développement du noyau. Les serveurs web ont toujours été un problème intéressant, car c'est la seule application réelle qui demande beaucoup au noyau. D'un certain côté, les serveurs web ont représenté un certain danger de mon point de vue car je reçois tant de commentaires de la part de la communauté utilisant Linux comme plate-forme de serveur web que j'aurais facilement pu finir par optimiser uniquement pour ce type d'applications. Je dois garder à l'esprit le fait que les serveurs web ne sont pas la seule application intéressante.

Bien entendu, Linux n'est pas utilisé à son plein potentiel, même par les serveurs web d'aujourd'hui. Apache lui-même ne procède pas Comme Il Faut en ce qui concerne les threads, par exemple. Ce type d'optimisation a été ralenti par les limites des réseaux. En ce moment, vous pouvez saturer les réseaux à dix mégabits par seconde si facilement qu'il n'y a aucune raison d'optimiser davantage. La seule manière de ne pas saturer les réseaux à dix mégabits par seconde est d'avoir des tas de scripts CGI fort consommateurs de ressources. Mais ce n'est pas quelque chose que le noyau peut éviter. Ce que le noyau pourrait éventuellement faire, c'est de répondre directement aux requêtes concernant les pages statiques et transmettre les requêtes plus compliquées à Apache. Une fois que les réseaux plus rapides seront devenus plus répandus, ce sera plus intéressant. Mais nous n'avons pour le moment pas suffisamment de matériel pour le tester et le développer.

La leçon que donnent toutes ces futures directions possibles est que je veux que Linux soit à la pointe du progrès, et même un peu au-delà, parce que ce qui est à la pointe aujourd'hui est ce qui sera dans nos ordinateurs personnels demain.

Mais les développements les plus attrayants pour Linux auront lieu dans l'espace utilisateur, et non pas dans l'espace noyau. Les changements du noyau sembleront insignifiants par rapport à ce qui arrivera en dehors du système. De ce point de vue, savoir ce que le noyau Linux sera n'est pas aussi intéressant que de savoir quelles seront les fonctionnalités de la Red Hat 17.5 ou quelle sera la situation de Wine (l'émulateur MS-Windows) dans quelques années.

Je m'attends à ce que, dans une quinzaine d'années, une autre personne débarque et dise : « je peux faire tout ce que Linux fait, tout en étant plus concis, parce que mon système ne sera pas alourdi par vingt ans d'héritages ». Ils diront que Linux était conçu pour le 386 et que les nouveaux processeurs effectuent les opérations vraiment intéressantes d'une manière différente. Laissons tomber ce vieux Linux. J'ai procédé ainsi lorsque j'ai commencé à développer Linux. À l'avenir, ils pourront regarder notre code, utiliser nos interfaces, fournir une compatibilité binaire. Et si tout cela se produit, je serai heureux.

Chapitre 9

En faire cadeau

9.1 Red Hat

Ou comment Red Hat Software a par hasard pris connaissance d'un nouveau modèle économique et a contribué à l'amélioration d'une industrie.

Le fait que je sois le fondateur de l'une des sociétés commerciales les plus importantes dans le domaine du logiciel libre ne confère pas à mes propos le statut d'élément utile à une recherche ou à une analyse intellectuelle objective. Le lecteur sceptique ne considérera donc pas le présent comme un document de référence sur ce sujet, mais simplement comme un ensemble d'anecdotes intéressantes, instructives, ou simplement étranges qui jouèrent sur le développement de Red Hat Software, Inc.

9.2 D'où vient Red Hat ?

Aux premiers jours du système Linux (1993), nous étions une petite société de distribution de logiciels. Nous proposions à bas prix des applications, des livres, et des CD-ROM Unix de vendeurs comme Walnut Creek et Infomagic. En plus de leur offre classique Unix, ceux-ci commencèrent à fournir une nouvelle ligne de produits : des CD-ROM Linux. Ils devinrent nos meilleures ventes. Quand nous demandions d'où venaient toutes ces choses pour Linux, on nous répondait à peu près : « Cela vient de programmeurs qui, en fonction de leurs compétences, répondent aux besoins des utilisateurs. »

Si la chute du Mur de Berlin nous a appris quelque chose, c'est bien que le socialisme seul n'est pas un modèle économique viable. Lorsque l'on néglige les slogans pleins d'espoir, il faut bien constater que les activités humaines ne se multiplient pas d'elles-mêmes sans un bon modèle économique dirigeant l'effort. Linux semble ne pas avoir un tel modèle. Nous en avons donc conclu qu'il n'était qu'un gros coup de chance. Un coup de chance qui générait suffisamment d'argent pour garder notre petite affaire, ainsi qu'un certain nombre d'autres, dans le vert ; mais c'était tout de même un coup de chance.

Quoi qu'il en soit, au lieu de s'écrouler, ce système d'exploitation étrange appelé Linux a continué de s'améliorer. Son nombre d'utilisateurs augmentait régulièrement et les applications portées gagnaient en sophistication.

Nous avons donc commencé à étudier plus attentivement son développement, parlé aux développeurs clés et à d'obscurs utilisateurs. Nous percevions peu à peu un modèle économique solide bien qu'inhabituel.

Ce modèle économique prouvait son efficacité. Et plus important, nos ventes de Linux, comparées à celles des autres Unix, étaient suffisantes pour nous convaincre qu'une vraie nouveauté technique, à l'avenir prometteur, émergeait. Durant l'automne 1994, nous avons cherché des produits Linux que nous pourrions vendre à CompUSA et à d'autres revendeurs. Marc Ewing et moi-même nous sommes donc associés pour créer Red Hat Software, Inc. en janvier 1995, et le reste de ce chapitre traite de nos tâtonnements lors du développement d'un plan commercial compatible avec cet étrange modèle économique qui produisait un système d'exploitation remarquable, apportant des avantages à nos clients et des profits à nos actionnaires.

Le rôle de Red hat est de travailler avec toutes les équipes de développement sur l'Internet afin de rassembler environ quatre cents outils en un système d'ex-

ploitation utilisable. Nous travaillons comme une usine d'assemblage de voitures car testons le produit fini et fournissons un service d'assistance et des services aux utilisateurs du système d'exploitation Red Hat Linux.

La satisfaction du besoin de nos clients était alors, et demeure aujourd'hui, la seule « source de profits » de notre stratégie commerciale. Nos clients, compétents sur le plan technique, souhaitent conserver la maîtrise du système d'exploitation qu'ils utilisent, et nous leur proposons de bénéficier d'un logiciel librement redistribuable (le code source et une licence libre).

9.3 Comment gagnez-vous de l'argent avec du logiciel libre ?

Cette question laisse penser qu'il est facile, ou au moins plus facile, de faire de l'argent en vendant des logiciels propriétaires uniquement sous forme binaire.

C'est une erreur. La plupart des éditeurs de logiciel libre ou propriétaire échouent. Jusqu'à très récemment tous appartenaient à la famille « binaires propriétaires seulement », on peut donc dire sans prendre de risque que le modèle de la propriété intellectuelle du développement et du commerce de logiciel est un moyen très difficile de gagner sa vie. Bien sûr, il en était de même pour les orpailleurs actifs pendant la ruée vers l'or du dix-neuvième siècle. Mais lorsqu'un éditeur de logiciel trouve un bon filon il gagne énormément d'argent, tout comme au temps des anciennes ruées vers l'or, donc beaucoup sont donc prêts à en assumer les risques pour bénéficier de l'occasion de « trouver de l'or ».

Personne ne s'attend à gagner facilement de l'argent avec du logiciel libre. Même si c'est un défi, ce n'est pas nécessairement plus difficile qu'avec du logiciel propriétaire. En fait, on gagne de l'argent avec du logiciel libre exactement de la même manière qu'avec du logiciel propriétaire : en construisant un superbe produit, en le commercialisant avec compétence et imagination, en cherchant des clients, et en bâtissant ainsi une marque synonyme de qualité et de service.

Vendre avec compétence et imagination, particulièrement sur des marchés encombrés de concurrents, implique d'offrir à ses clients des solutions que les autres ne peuvent ou ne veulent pas offrir. À ce niveau, le logiciel libre (Open Source) n'est pas un handicap mais bien un avantage. Le modèle de développement Open Source conduit à un logiciel stable, souple, et hautement adaptable à des besoins spécifiques. L'éditeur de logiciel Open Source dispose donc d'emblée d'un produit de qualité et l'astuce consiste à inventer un moyen efficace de gagner de l'argent tout en laissant au client les avantages de ce type de programmes.

Inventer de nouveaux modèles économiques n'est pas une tâche facile, et le mode d'innovation découvert par Red Hat ne s'applique certainement pas à toutes les entreprises et à tous les produits. Certains concernent toutefois de nombreux éditeurs de logiciels, et beaucoup d'entreprises Open Source.

Plusieurs sociétés abordent le marché avec une approche Open Source tronquée car elles adoptent le plus souvent une licence autorisant une libre distribution de leur logiciel si l'utilisateur ne l'emploie pas à des fins commerciales. Les autres doivent dans ce cas payer à l'éditeur un droit de licence ou des royalties. On définit l'Open Source comme un logiciel qui comprend son code source et une licence libre et gratuite — les entreprises partiellement Open Source fournissent

le code source, mais pas de licence libre.

Souvenez-vous, la diffusion et la croissance de la part de marché du logiciel libre ne font que commencer. Si on ne gagne pas d'argent aujourd'hui, c'est peut-être parce que le marché de notre produit est encore restreint. Bien que l'on soit satisfait de la croissance du système Linux, car les estimations lui accordent aujourd'hui (1998) 10 millions d'utilisateurs, il faut se souvenir que 230 millions d'utilisateurs emploient MS-DOS/MS-Windows.

9.4 Nous vendons des produits de grande consommation

Les éditeurs détiennent les droits découlant de la propriété intellectuelle liée à leurs produits, mais ce n'est pas notre cas. Ils insistent sur le fait que leur plus grand bien est cette propriété, conférée par le fait qu'ils conservent par-devers eux les codes sources de leurs logiciels. On peut donc affirmer que Red Hat n'est pas un éditeur de logiciels car nous ne fournissons pas de licence d'exploitation du fruit de notre propriété intellectuelle. Cette approche n'avantagerait en effet pas nos clients, employés et actionnaires. La question devient donc : dans quel secteur commercial évoluons-nous ?

Pour trouver une réponse, il fallait analyser d'autres secteurs d'activité économique et essayer d'y découvrir une ressemblance. Une industrie où les ingrédients de base sont gratuits, ou tout au moins librement disponibles. Nous avons regardé du côté du domaine juridique car nul ne peut déposer ou breveter des arguments légaux. Si un avocat gagne une affaire devant la Court Suprême, les autres avocats peuvent ensuite utiliser ses arguments sans devoir en obtenir la permission car ils sont tombés dans le domaine public.

L'industrie automobile, elle aussi, présente quelques similitudes car on peut se procurer les pièces auprès d'un grand nombre de fournisseurs. Personne ne conduit une voiture, nous conduisons des Honda, des Ford, ou bien n'importe lequel des quelques centaines de modèles différents, assemblés à partir de pièces communes disponibles. Quelques personnes sauraient assembler leurs propres voitures et, parmi elles, peu en ont le temps ou l'envie. L'assemblage et le service forment le noyau du modèle économique de l'industrie automobile.

L'étude de l'industrie des produits de grande consommation nous montra que toutes les grandes entreprises érigent leur stratégie commerciale sur le renforcement de l'image de marque. L'eau minérale (Perrier, Evian), la lessive (Ariel), ou la sauce tomate (Heinz) en sont autant d'exemples patents. Ces marques doivent être synonymes de qualité, de cohérence, et de fiabilité. Cela nous inspira.

Le ketchup n'est rien d'autre que de la sauce tomate assaisonnée. On peut facilement faire quelque chose qui ressemble beaucoup et a presque le même goût que le ketchup Heinz, sans pour autant détourner un copyright. Les ingrédients sont tous librement redistribuables : des tomates, du vinaigre, du sel, et des épices. Pourquoi les clients n'élaborent-ils pas leur propre ketchup dans leurs cuisines ? Pourquoi Heinz détient-il 80% du marché ?

Tout simplement parce que le produit industriel est moins cher et beaucoup plus pratique. Mais l'aspect pratique n'explique pas tout, sinon Heinz, Amora, et Del Monte se partageraient équitablement le marché, puisqu'ils offrent à peu de choses près le même produit. Mais Heinz détient 80% du marché...

Heinz ne détient pas cette part du marché parce que le goût de son ketchup est plus convaincant. Cent habitants d'un pays du Tiers-Monde qui n'ont jamais goûté au ketchup nous apprendront tout d'abord que les gens n'aiment pas le ketchup, et aussi qu'ils détestent tous les ketchups.

Heinz détient 80% du marché du ketchup car il a été capable de définir le goût du ketchup dans l'esprit des consommateurs. L'efficacité de leur approche est telle qu'en tant que clients nous pensons qu'un ketchup qui ne sort pas de la bouteille est, d'une certaine façon, meilleur qu'un autre qui, lui, coule facilement !

C'était la solution pour Red Hat : offrir du confort, de la qualité, et, ce qui en fait est le plus important, aider à concrétiser dans l'esprit de nos clients ce qu'un système d'exploitation peut être. Red Hat, de même, peut fournir et maintenir un produit cohérent et de grande qualité afin de bénéficier d'une magnifique occasion d'établir une marque que les clients du système Linux préféreront, tout simplement.

Mais comment concilier notre besoin d'augmenter le nombre d'utilisateurs de Linux avec celui de nous assurer qu'ils choisissent Red Hat ? Nous avons étudié le mode de fonctionnement des industriels réalisant des bénéfices grâce aux activités des autres participants, et non malgré eux.

Chacun peut boire de l'eau, dans la plupart des pays industrialisés, en ouvrant le robinet le plus proche. Pourquoi Evian vend-il donc plusieurs millions de dollars d'eau minérale ? Cela provient d'une peur, en grande partie irrationnelle, qui nous empêche d'avoir confiance en l'eau coulant du robinet.

Pour la même raison, beaucoup de clients préfèrent acheter la version « Officielle » de Linux Red Hat, facturée 50 dollars, alors qu'ils pourraient la télécharger gratuitement ou acheter une copie non officielle sur un CD-ROM vendu 2 dollars. Evian dispose, à ce titre, d'un avantage certain, car tout le monde boit de l'eau. Nous devons donc encore accompagner beaucoup de consommateurs vers Linux afin d'augmenter la taille du marché.

Notre part de marché importe donc moins que sa taille. Evian profite de toute augmentation de la demande des acheteurs d'eau en bouteille, même si beaucoup achètent tout d'abord une bouteille proposée par une autre marque. Red Hat, comme Evian, profite du bon travail d'autres fournisseurs de Linux, quand ils développent le marché du produit. Le nombre de clients potentiels pour la version Red Hat augmente avec celui des utilisateurs de Linux.

Le pouvoir des marques se transpose très efficacement au commerce des techniques. Nous en avons eu la preuve avec les investisseurs qui ont récemment investi dans plusieurs sociétés de logiciels Open Source dont le dénominateur commun est qu'elles jouissent d'une grande notoriété, et sont reconnues comme proposant des produits de qualité. En d'autres termes, elles ont établi une marque.

9.5 Attrait stratégique de ce modèle pour l'industrie informatique

La plus grande partie de la gestion d'une marque consiste en son positionnement sur le marché. Étudiez les défis auxquels doit faire face un nouveau système d'exploitation afin de gagner une part non négligeable d'un marché sa-

turé et dominé par un favori bien connu, vendu par une société au marketing brillant. Présenter de façon adéquate un produit rival est crucial pour vaincre un concurrent.

Linux tient naturellement fort bien ce rôle. Le principal reproche adressé au leader du marché est son monopole dans la fourniture de systèmes. Un nouveau concurrent doit donc permettre à l'utilisateur de garder le contrôle de sa plateforme, donc ne pas devenir un autre système propriétaire binaire, avec lequel l'éditeur obtiendrait une position dominante sur le marché, puisque c'est ce que les clients critiquent actuellement.

Linux, tout bien considéré, n'est pas vraiment un système. mais plutôt une collection de composants Open Source, de la même manière que le mot « automobile » décrit une industrie plus que l'objet que nous conduisons sur l'autoroute. On ne conduit pas des automobiles, mais des Ford Taurus ou des Honda Accord. Red Hat est une sorte d'usine d'assemblage d'un système fourni par l'industrie du système d'exploitation libre. Red Hat a gagné son pari lorsque les clients ne se croient pas acheteurs d'un système d'exploitation, ou même de Linux, mais surtout et avant tout clients de Red Hat.

Honda achète des pneus à Michelin, des airbags à TRW, de la peinture à Dupont, et assemble ces diverses pièces en une Accord livrée avec une certification, des garanties, et un réseau de réparateurs Honda et indépendants.

Red Hat prend les compilateurs de Cygnus™, le serveur web Apache, le système X Window du X Consortium (développé avec l'aide de Digital, HP, IBM et Sun, entre autres), et assemble tout cela en un système d'exploitation Red Hat Linux que l'on peut certifier, garantir, et qui a gagné de nombreuses récompenses.

Le travail de Red Hat s'apparente à celui des industriels de l'automobile. Red Hat prend ce que l'on considère comme les meilleurs composants Open Source disponibles afin de réaliser le meilleur système possible. Mais nul ne dirige le développement de ce dernier. Un client de Red Hat qui ne partage pas notre préférence pour Sendmail et souhaite utiliser Qmail ou une autre solution en lieu et place reste libre de le faire. De même, quelqu'un qui achète une Ford Taurus peut vouloir installer dans son moteur un collecteur d'échappement aux performances supérieures. Le propriétaire de Taurus reste libre d'ouvrir le capot et de décider de ce que contient sa voiture. L'utilisateur de Red Hat gère lui aussi son système parce qu'il détient une autorisation de modifier le code source.

On ne peut rivaliser avec un monopole en adoptant ses règles. Ses ressources, ses canaux de distribution, les moyens consacrés à son service de recherche et développement constituent autant de point forts. On rivalise en changeant les règles du jeu pour favoriser ses propres points forts.

À la fin du dix-neuvième siècle, les plus grandes compagnies de chemin de fer américaines détenaient un monopole de fait sur les transports reliant les grandes villes. Ces dernières, comme Chicago, se sont étendues autour de terminaux centraux de voies ferrés appartenant aux compagnies de chemin de fer.

Ces monopoles ne tombèrent pas parce que quelqu'un déploya de nouvelles voies ferrées et adopta des tarifs plus avantageux pour les clients, mais lors de la construction du système d'autoroutes reliant les états et grâce aux avantages de la livraison par le cheminement porte-à-porte des transporteurs employant des camions, plus adéquat que le point à point du chemin de fer.

De nos jours, les éditeurs de systèmes propriétaires détiennent les clés d'une technique qui présente une forte analogie avec le système de chemin de fer. Les

interfaces de programmation (Application Programming Interface, API) d'un système propriétaire sont comme les voies et les horaires du chemin de fer. Leurs détenteurs établissent librement leurs tarifs. Ils peuvent aussi adapter le « chemin » des APIs aux besoins de leurs propres applications. Ils limitent l'accès au code source sur lequel doivent fonctionner leurs applications mais aussi celles des éditeurs indépendants de logiciels (Independent Software Vendors, ISV), ce qui leur offre un avantage commercial décisif.

Pour sortir du carcan de ce modèle, les ISV ont besoin d'un système dont l'éditeur n'a pas le contrôle, dont le fournisseur n'endosse que la maintenance, et pour lequel ils commercialiseront leurs applications en restant assurés que l'éditeur du système ne deviendra jamais un concurrent sérieux. Le monde du logiciel commence à reconnaître les vertus de ce modèle. Ce raisonnement a conduit Corel à porter WordPerfect sous Linux, Oracle à proposer une version de son système de gestion de base de données, IBM à soutenir Apache.

Le contrôle laissé aux utilisateurs d'un système Open Source assoit sa supériorité sur ses concurrents propriétaires, dont seuls les binaires sont disponibles et dont les éditeurs, aux investissements énormes, ne peuvent essayer de nous imiter puisque nous ne générons qu'une fraction du revenu par utilisateur dont ils dépendent.

Si un groupe suffisamment étoffé d'utilisateurs adopte notre modèle économique, les éditeurs de systèmes non libres devront réagir. Mais cela ne sera pas le cas avant quelques années encore. S'ils réagissent en « libérant » leur code source, comme Netscape libéra celui de Navigator, il en résultera de meilleurs produits au coût beaucoup moins élevé. Ce résultat de nos efforts bénéficiera à toute l'industrie, mais Red Hat ne se propose pas de s'en tenir là.

Le cas du Fermilab illustre bien l'importance du fait que les utilisateurs de Linux détiennent les clés de leurs royaumes. Ce laboratoire de recherche dispose d'un gros accélérateur de particules installé dans la région de Chicago, où plus d'un millier de physiciens adaptent des techniques dernier cri. Linux présente l'avantage de pouvoir fonctionner en grappes (clusters), pour constituer des super-ordinateurs massivement parallèles. Fermilab a besoin de cela, car il se propose d'augmenter les performances de son accélérateur : il s'attend ainsi à devoir analyser pratiquement 10 fois plus de données par seconde qu'il ne le faisait jusqu'alors. Son budget ne lui permettrait tout simplement pas d'acquérir les super-ordinateurs offrant la puissance de calcul nécessaire.

Pour cette raison, parmi d'autres, Fermilab souhaitait employer un logiciel Open Source. Il a constaté que Red Hat Linux était un des produits de ce type les plus populaires et il nous a donc contacté. En fait, il nous a appelé six fois durant les quatre mois de la phase de sélection du système, et nous n'avons même pas répondu une seule fois à leurs demandes. Quoi qu'il en soit, le résultat de son étude a été de choisir Red Hat Linux en tant que système officiel employé. La morale de cette histoire est que

1. nous devons apprendre à mieux répondre au téléphone (nous l'avons fait)
2. Fermilab a pu percevoir que notre modèle commercial lui ménageait les libertés nécessaires, indépendamment de notre capacité à les assister.

Le système Linux présente des avantages pour les grandes organisations consommatrices d'ordinateurs et les grands fournisseurs de produits informatiques (ISV). Il ne souffre pas des grandes limitations des systèmes propriétaires actuellement disponibles uniquement sous forme binaire. Une gestion attentive de la marque Red Hat Linux parmi les distributions de Linux et un positionne-

ment soigné de Linux sur le marché des systèmes non propriétaires permettent à Red Hat de croître et de connaître le succès.

9.6 Licences, Open Source, et logiciel libre

Les avantages de Linux ne dépendent pas de sa grande fiabilité, de sa facilité d'utilisation, de sa robustesse, ni des outils inclus mais dans la maîtrise que l'on en a du fait de deux caractéristiques qui lui sont propres, à savoir qu'il est fourni avec le code source complet, et que l'on peut utiliser ce code source pour ce que l'on veut, sans même demander une autorisation.

La NASA, qui propulse des gens dans l'espace, a une expression : « un logiciel n'en est pas un sans son code source. »

La haute fiabilité ne satisfait pas, à elle seule, les ingénieurs ce cette agence. Une fiabilité extrêmement élevée ne les contentera pas davantage. La NASA a besoin d'une fiabilité absolue. Elle ne peuvent pas se permettre de subir un « écran bleu de la mort » avec douze âmes confiantes propulsées à deux milles kilomètres par heure autour de la Terre, dont les vies dépendent de son système.

La NASA a besoin d'avoir accès au code source du logiciel pour construire ces systèmes. Et elle a besoin de l'autorisation de le modifier pour l'adapter à ses besoins. Je dois admettre qu'il n'est peut-être pas nécessaire de rendre le système de gestion d'un cabinet de dentiste, servant à facturer le détartrage annuel des patients, aussi fiable que celui dont dépendent les astronautes de la NASA, mais le principe reste le même.

Nos utilisateurs peuvent modifier le produit pour l'ajuster aux besoins de l'application qu'ils mettent en place. C'est la valeur ajoutée que Red Hat offre à ses clients, une offre qu'aucun de nos rivaux, bien plus importants, ne veut, ou ne peut, formuler.

Cette valeur ajoutée renverse toutes les notions usuelles de propriété intellectuelle. La licence Red Hat, loin d'emprisonner les utilisateurs et de les éloigner du code source, intègre l'idée d'accès et de contrôle du code source. Quelle licence permettrait d'assurer cela ? Des personnes raisonnables de la communauté Open Source peuvent avoir et ont des avis différents sur la réponse. Mais chez Red Hat, nous avons nos propres positions sur le sujet. Les voici :

La General Public License (GPL), de la Free Software Foundation est dans l'esprit de l'Open Source et, parce qu'elle garantit que les modifications et améliorations apportées au système restent publiques, est très efficace pour gérer un projet de développement coopératif.

Notre définition de l'« efficacité » vient des vieux jours du développement d'Unix. Avant 1984, ATT partageait le code source du système Unix avec n'importe quelle équipe capable de les aider à l'améliorer. Quand ATT s'est scindé, les divisions en résultant n'étaient plus limitées à être des opérateurs de télécommunications. Ils ont alors décidé d'essayer de gagner de l'argent en vendant des licences du système d'exploitation Unix. Toutes les universités et les groupes de recherches qui avaient aidé à la construction d'Unix acquittèrent dès lors des factures afin de pouvoir l'employer alors même qu'ils avaient participé à son développement. Ils étaient furieux mais impuissants, car ATT possédait le copyright sur Unix. Ces autres équipes de développement ont poussé ATT à la discrétion.

Notre préoccupation est la même. Si Red Hat développe une innovation que nos concurrents peuvent utiliser, le moins que nous puissions demander est que nos équipes de développement disposent de la même façon des leurs. La GPL est la licence la plus efficace pour assurer que la coopération forcée entre les membres des différentes équipes continue malgré la concurrence du moment.

Gardons à l'esprit que la modularité du système Linux est l'une de ses grandes forces. Une version de Red Hat Linux abrite plus de 435 outils distincts. Le choix d'une licence a donc aussi un aspect pratique. Si elle permet à Red Hat de fournir un logiciel mais pas d'y apporter des modifications, cela crée des problèmes car les utilisateurs ne peuvent plus corriger ou adapter le logiciel pour leurs besoins. Une licence moins restrictive, qui oblige l'utilisateur à demander une autorisation à l'auteur avant de modifier un logiciel, contraint encore trop Red Hat et nos clients. Devoir demander l'autorisation de modifier à peut-être 435 auteurs ou équipes de développement différents n'est tout simplement pas pratique.

Mais nous ne sommes pas des fanatiques des licences. Nous nous satisfaisons d'une licence qui nous accorde un contrôle sur le logiciel utilisé car cela nous permet à notre tour d'offrir les avantages correspondants à nos clients et utilisateurs, qu'ils soient ingénieurs de la NASA ou programmeurs d'applications travaillant sur le système de facturation d'un cabinet dentaire.

9.7 Le moteur économique du développement de logiciel Open Source

Les histoires intéressantes sur les origines de Linux démontrent la puissance du modèle économique de son développement.

La communauté Open Source a dû dépasser le stéréotype du hacker hobbyiste. Selon ce stéréotype, Linux, par exemple, est développé par des hackers de quatorze ans, dans leurs chambres. On a ici un exemple de la peur, de l'incertitude et du doute (Fear, Uncertainty and Doubt, FUD), distillés et imposés à l'industrie du logiciel par les éditeurs de systèmes propriétaires. Après tout, qui pourrait faire confiance, pour son application critique, à un logiciel écrit par un adolescent pendant ses loisirs ?

La réalité, bien entendu, est très différente. Même si le « hacker solitaire » a toujours une grande valeur et une part importante dans le processus de développement, de tels programmeurs n'ont réalisé qu'une faible part du code qui constitue le système Linux. Le créateur de Linux, Linus Torvalds, a commencé à travailler sur Linux alors qu'il était étudiant, et la majeure partie du code est écrite par des développeurs professionnels travaillant dans les plus grandes organisations de développement, d'ingénierie et de recherche.

En voici quelques exemples : les compilateurs GNU C et C++ sont fournis par Cygnus Solutions Inc.™ (Sunnyvale, Californie), le système X Window par le X Consortium (avec l'aide d'IBM, HP, Digital, et Sun). Un bon nombre des pilotes Ethernet sont maintenant sous la responsabilité d'ingénieurs de la NASA. Les pilotes de périphériques proviennent dorénavant assez souvent des constructeurs des périphériques eux-mêmes. Bref, le mode de développement d'un logiciel Open Source ne diffère guère de celui d'un logiciel conventionnel, et le talent impliqué est dans l'ensemble le même.

Grant Guenther, à l'époque membre de l'équipe de développement du serveur de bases de données d'Empress Software, voulait que ses collaborateurs puissent travailler sur leurs projets à domicile. Ils avaient besoin d'une méthode sûre pour transférer de gros fichiers de leurs bureaux à leurs domiciles et réciproquement. Ils utilisaient Linux sur des PC équipés de lecteurs Zip, mais à cette époque (1996), Linux exploitait mal ces périphériques.

Donc Grant a eu le choix : abandonner la solution Linux et acheter une solution propriétaire plus chère, ou arrêter ce qu'il était en train de faire et passer quelques jours à écrire un pilote décent pour le lecteur Zip. Il a choisi de le développer, et il a collaboré avec d'autres utilisateurs de ces lecteurs réunis grâce à l'Internet, qui testèrent et améliorèrent son logiciel pilote (driver).

Tâchons de déterminer ce qu'un éditeur de logiciels aurait payé à Empress et Grant pour cela. On peut sans risque l'estimer à plusieurs dizaines de milliers de dollars, et cependant Grant a choisi de « faire cadeau » de son travail. En retour, au lieu de l'argent, il a bénéficié d'une excellente solution à son problème de travail à domicile pour les programmeurs d'Empress, pour une fraction du prix des autres options possibles. Voilà le genre de proposition gagnant-gagnant offerte par les modèles coopératifs comme celui du développement Open Source.

9.8 Des avantages uniques

Il est facile de confondre caractéristiques et avantages. Les modèles Open Source en général et Linux en particulier présentent sans aucun doute des caractéristiques uniques, et certains pensent qu'elles sont la raison pour laquelle tout le monde adopte Linux avec un tel enthousiasme. Comme des centaines de directeurs de systèmes d'information (Management Information System, MIS) me l'ont fait remarquer, « pourquoi quelqu'un souhaiterait-il disposer du code source de son système d'exploitation ? ». Le fait est que personne ne veut du code source. Personne n'a besoin d'une licence de logiciel libre. Ce sont juste les caractéristiques du système. Mais une caractéristique n'est pas forcément un avantage. Donc quel est l'avantage associé à cette caractéristique ?

Pour le comprendre, nous ne devons pas négliger que la solution la plus élégante s'impose rarement, même dans les marchés les plus techniques (et cela désespère les informaticiens). Construire la meilleure souricière ne vous assure pas le succès. Linux n'obtiendra pas des succès parce qu'il peut être installé sur une machine avec moins de mémoire que les autres systèmes, ou parce qu'il coûte moins cher, ou parce qu'il est plus fiable. Ce ne sont que des caractéristiques qui font de Linux une meilleure souricière que NT ou OS/2.

Les forces qui conduiront peut-être Linux au succès ou à l'échec uvrent à un autre niveau. En fait, ces facteurs sont généralement regroupés sous le terme de « position sur le marché ». Comme un cadre supérieur de Lotus nous l'a demandé récemment, « avons-nous vraiment besoin d'un nouveau système ? » Linux ne réussira que s'il est plus qu'« un système de plus ». En d'autres termes, Linux représente-t-il un nouveau modèle pour le développement et le déploiement d'un système ou n'est-il qu'« un système de plus » ?

La réponse est : Linux et le mouvement Open Source dans son ensemble sont une révolution du mode de développement qui améliorera les systèmes informatiques en cours de conception et à venir.

L'Open Source est une caractéristique possible pour un code. L'avantage réside dans la maîtrise offerte à l'utilisateur. Toute entreprise souhaite disposer du contrôle sur son logiciel, et la caractéristique de l'Open Source reste la meilleure approche mise au point par l'industrie pour l'offrir.

9.9 Le grand défaut d'Unix

Certaines personnes affirment avec force que Linux se balkanisera comme tous les Unix. La réponse offre à mon sens la meilleure illustration du caractère très nouveau de l'approche adoptée par ses développeurs. Il existe aujourd'hui, en apparence, trente versions différentes du système Unix, pour la plupart incompatibles.

Mais les forces qui ont séparé les différents Unix travaillent maintenant à l'unification des versions de Linux.

La différence principale entre Unix et Linux ne réside pas dans le noyau, ou le serveur Apache, ou toute autre caractéristique de ce genre. C'est qu'Unix n'est qu'un système propriétaire de plus, dont seuls les binaires sont livrés, et qui reste fondé sur la propriété intellectuelle. Cela signifie que les éditeurs de ces Unix subissent des pressions commerciales à court terme, les invitant à conserver toute innovation ajoutée au système afin d'en réserver les bénéfices à leurs seuls clients. Avec le temps, les ajouts successifs de ces « innovations propriétaires » aux différentes versions d'Unix séparèrent peu à peu ces derniers, de façon parfois décisive. Les autres fournisseurs n'ont pas accès au code source de l'innovation et sa licence interdit de la reproduire, même si tous voudraient l'utiliser.

Pour Linux, tout fonctionne à l'envers. Si l'innovation proposée par un fournisseur de Linux devient populaire, les autres fournisseurs l'adopteront immédiatement, car ils disposent du code source correspondant, et sont autorisés à l'utiliser.

Tous les sceptiques, par exemple, prédirent la chute de Linux lors du débat de 1997 portant sur les anciennes bibliothèques C (libc) et la nouvelle version (glibc). Red Hat a adopté les nouvelles pour des raisons techniques. Certaines distributions populaires de Linux utilisèrent encore longtemps les anciennes versions. Le débat a fait rage pendant six mois. Pourtant, alors que 1998 touchait à sa fin, toutes les distributions importantes de Linux avaient opté ou annoncé qu'elles opteraient pour les plus récentes bibliothèques, plus stables, plus sûres, et plus performantes.

C'est une partie de la puissance de l'Open Source : il crée une sorte de pression unificatrice vers un point de référence commun (un standard ouvert, en fait) et cela fait tomber les barrières de la propriété intellectuelle, qui entraveraient sinon cette convergence.

9.10 À vous de choisir

À chaque fois qu'une nouvelle pratique révolutionnaire apparaît, des sceptiques prédisent sa chute inévitable et montrent du doigt tous les obstacles qu'elle devra surmonter. Des fanatiques prêchent que seules les implémentations les plus pures du nouveau modèle peuvent réussir. D'autres, comme nous,

se contentent de travailler, de tester, d'innover, et de l'utiliser lorsqu'elle s'avère plus efficace que l'ancienne pratique.

La naissance du PC illustre elle aussi le principal avantage de ce nouveau modèle. L'enthousiasme déclenché par la première spécification du PC IBM, publiée en 1981, ne devait rien à des critères d'ordre technique. Le premier PC intégrait une puce 8086 et 64 Ko (oui, Ko) de mémoire centrale. la mémoire totale utilisable ne pouvait dépasser 640Ko. Personne n'imaginait qu'une machine individuelle exigerait un jour davantage. Un magnétophone à cassettes sauvegardait les données.

La révolution du PC prit son essor parce que ses utilisateurs restaient libre de disposer comme ils le souhaitaient de leur plate-forme informatique. Ils pouvaient acheter leur premier PC chez IBM, leur deuxième machine chez Compaq, et leur troisième chez HP. Ils pouvaient acheter de la mémoire ou un disque dur auprès d'un fournisseur parmi une centaine, et une gamme presque infinie d'équipements périphériques satisfaisaient presque tous les besoins.

Ce nouveau modèle a introduit un très grand nombre d'incohérences, d'incompatibilités, et de confusions entre les techniques, les produits, et les fournisseurs. Mais comme tout le monde le sait maintenant, les clients adorent le choix et la possibilité de maîtriser l'équipement. Ils s'accommoderont pour cela d'une certaine dose de confusion et d'incohérence.

Notez aussi que le marché du matériel PC ne s'est pas fragmenté. Les spécifications sont généralement restées ouvertes, et une forte pression oblige les industriels à se conformer aux standards pour préserver l'interopérabilité. Aucun d'eux n'a une souricière suffisamment meilleure pour attirer les clients et ensuite les prendre en otages dans une solution propriétaire. Au lieu de cela, les innovations (les meilleures souricières) reviennent à la communauté dans son ensemble.

Le système Linux laisse le choix aux clients quant aux solutions techniques fournies avec leurs ordinateurs pour ce qui relève du système d'exploitation. Cela demande-t-il un tout nouveau niveau de responsabilité et d'expertise de la part de l'utilisateur ? Certainement.

Préféreront-ils, après avoir goûté au choix et à la liberté du nouveau modèle, revenir à l'ancienne approche les obligeant à faire confiance au fournisseur du système propriétaire ? C'est peu probable.

Les sceptiques continueront à chercher, et parfois à trouver, des problèmes sérieux liés à l'approche Linux. Mais les clients adorent le choix, et l'immense marché du développement de logiciel Open Source effectué via l'Internet les résoudra tôt ou tard.

Chapitre 10

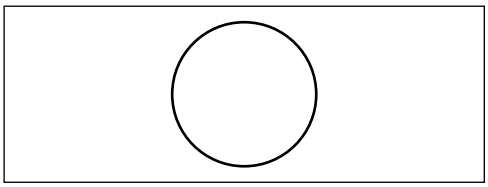
Diligence, Patience et Humilité

La communauté Perl a une prédilection pour les dictons. L'un d'entre eux est « Il y a plus d'une façon de faire ». C'est le cas en Perl. C'est le cas pour Perl lui-même. C'est aussi le cas pour la communauté du logiciel libre, comme vous pouvez le voir dans les textes qui composent ce volume. Je ne vais pas tout vous expliquer sur le logiciel libre et son fonctionnement. Mais je vais dire quelques mots sur Perl, où il en est, et vers où il se dirige.

Voici un autre dicton : Les trois grandes vertus de la programmation sont la paresse, l'impatience, et l'orgueil. Les grands programmeurs Perl chérissent ces vertus, et les développeurs de logiciel libre en font autant. Mais c'est d'autres vertus que je vais parler ici : la diligence, la patience, et l'humilité. Vous trouvez peut-être qu'elles semblent opposées aux précédentes, et vous avez bien raison. Et si vous ne croyez pas qu'une seule et même communauté puisse pratiquer des vertus opposées, c'est que vous n'avez pas passé assez de temps avec Perl. Après tout, il y a plus d'une façon de faire.

Les langues écrites sont probablement issues de l'impatience, ou de la paresse. Sans l'écriture, pour communiquer avec quelqu'un, il fallait l'avoir en face de soi, ou alors persuader quelqu'un d'autre de porter le message à votre place. Et le seul moyen de savoir ce qui avait été dit, était d'en garder le souvenir. L'écriture nous a donné des symboles, qui peuvent représenter des choses à condition que la communauté se mette d'accord sur les correspondances. Le langage nécessite donc un consensus. Il s'agit donc d'un ensemble de symboles qui unit une communauté. C'est en fait le cas pour la plupart des symboles.

Examinons à présent quelques symboles :

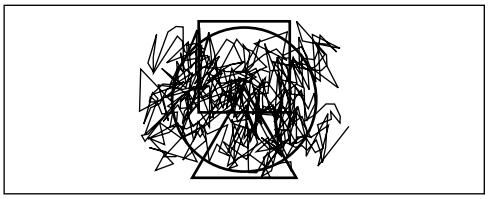


Observez-le de près. C'est un cercle. Un beau cercle, très joli. Très symétrique. Très simple.

Si vous êtes un réductionniste, vous trouverez que c'est n'est qu'un cercle, et rien d'autre. Les ultra-réductionnistes, quant à eux, n'y verront qu'un amas de photons. Suis-je clair ?

Si vous n'êtes pas un réductionniste, le cercle que vous voyez ici n'est pas une entité isolée. Il n'existe que par rapport à de nombreuses autres choses, et reçoit son sens d'elles. Pour comprendre ce simple cercle, il faut d'abord comprendre son contexte, c'est-à-dire comprendre un peu la réalité qui l'entoure.

Voici donc une représentation de la réalité :



Comme tout le monde le sait, la réalité est désordonnée.

Cette image représente beaucoup de choses. C'est une image de molécules d'air qui se baladent, une image de l'économie, une image des relations humaines

dans une salle, une image de ce à quoi ressemble une langue humaine typique, une image du système informatique de votre entreprise, une image du World Wide Web. C'est une image du chaos et de la complexité.

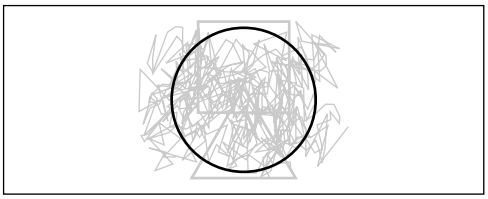
C'est aussi une image de l'organisation de Perl, puisque Perl prend pour modèle les langues humaines, et que les langues humaines sont complexes, justement parce qu'elles doivent faire face à la réalité.

Nous avons tous besoin de faire face à la réalité, d'une façon ou d'une autre. Pour cela, nous simplifions souvent trop !

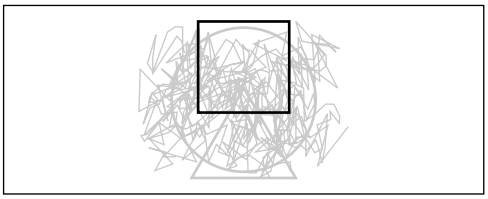
Les anciens simplifièrent trop. Convaincus que Dieu n'avait créé que des cercles et des sphères, ils croyaient qu'il préférerait toujours la simplicité à la complexité. Quand ils ont découvert que la réalité était plus complexe qu'ils ne l'avaient cru, ils ont caché la complexité derrière un rideau d'épicycles. Et ce faisant, ils ont créé de la complexité superflue. Là est la distinction importante : l'univers est complexe, mais pas inutilement.

Tout semble suggérer qu'aujourd'hui encore on simplifie trop. Certains choisissent de trop simplifier leur cosmologie. D'autres leur théologie. Et nombre d'informaticiens simplifient trop les langages qu'ils créent, et cachent la complexité de l'univers derrière le rideau du programmeur.

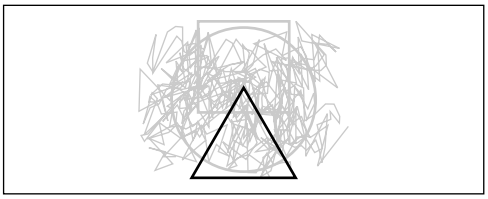
Chercher des motifs dans le bruit est une tendance humaine naturelle, mais en cherchant ces motifs, nous en voyons parfois qui ne sont pas vraiment là. Ceci ne signifie pas qu'il n'y ait pas de motifs. Si on nous donne une baguette magique pour supprimer le bruit, la régularité apparaît à l'il nu. Avec un coup de baguette magique pour supprimer le bruit, Abracadabra voici la forme du big bang, et des étoiles, et des bulles de savon.



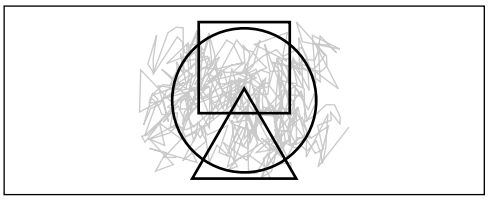
Voici la forme de la dimensionalité, des cristaux de sel, et des vides entre les troncs d'arbre.



Voici la forme d'un nid de fourmis, ou d'un arbre de Noël, ou la forme d'une trinité :



Et, bien sûr, une fois que vous avez vu que les motifs sont là, vous arriverez à les distinguer sans l'aide des couleurs (ou du contraste) :



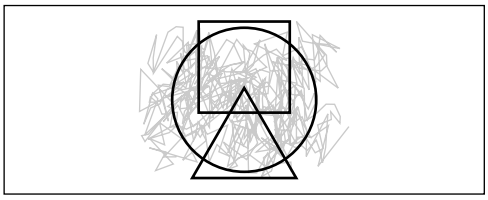
Nos cerveaux sont doués pour ça.

À ce point-là, vous vous demandez peut-être quel est le rapport entre tout ceci et Perl. Le fait est que vos cerveaux sont faits pour programmer en Perl. Vous avez un désir profond de rendre simple ce qui est complexe, et Perl n'est rien d'autre qu'un outil servant à cela. Tout comme j'utilise en ce moment même le français pour tenter de simplifier la réalité. Le français est utile pour cela, parce qu'il est désordonné aussi.

C'est important, et aussi un peu difficile à comprendre. Le français est utile parce qu'il est désordonné. Comme il est désordonné, il s'applique bien à l'espace des problèmes, lui aussi désordonné, que l'on appelle réalité. De façon similaire, Perl a été construit pour être désordonné (mais de la façon la plus agréable possible).

Permettez-moi d'insister sur ce point contraire à l'intuition. Si vous avez une formation d'ingénieur, on vous a rabâché que la première qualité d'un travail est sa simplicité. On nous apprend à admirer un pont suspendu plus qu'un tréteau de voie ferrée. On nous apprend à admirer la simplicité et la beauté. Ce n'est pas mauvais en soi ; moi aussi j'aime bien les cercles.

La complexité n'est pourtant pas toujours l'ennemi. Le plus important n'est pas la simplicité ni la complexité, mais comment on relie les deux.



Toute construction nécessite une certaine mesure de complexité. On dit que la fusée SaturnV comptait sept millions de pièces, qui devaient toutes fonctionner. Mais ce n'est pas tout-à-fait vrai. Un grand nombre de ces pièces étaient redondantes. Et cette redondance était absolument nécessaire pour qu'il soit possible d'expédier quelqu'un sur la Lune en 1969. Si certaines de ces pièces avaient pour fonction d'être redondantes, il fallait quand même qu'elles fassent leur boulot, pour ainsi dire.

Nous nous trompons quand nous disons « C'est redondant » en voulant dire « C'est inutile ». La redondance n'est pas toujours « redondante », que l'on parle de fusées ou de langues humaines ou de langages de programmation. En fait, la simplicité est bien souvent un obstacle à la réussite.

Supposons que je veuille conquérir le monde. La simplicité dit que je devrais le faire tout seul. Mais la réalité impose que j'ai absolument besoin de votre aide pour cela, et que vous êtes tous très complexes. Et je trouve que c'est une bonne

chose Vos inter-relations sont encore plus complexes, et je trouve aussi que c'est une bonne chose Parfois il y a des bogues, mais il est possible de déboguer des relations.

J'ai argumenté que la complexité peut tout aussi bien être inutile qu'utile. Voyons encore un exemple de complexité utile :

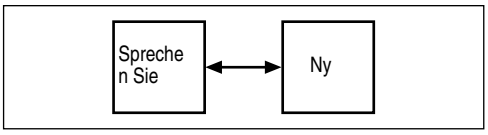
Vous avez probablement des préjugés en faveur des systèmes d'écriture occidentaux, et à l'encontre des systèmes idéographiques, inutilement complexes. Vous trouvez peut-être que cette image est tout aussi compliquée que la précédente. Mais, encore une fois, il s'agit d'un compromis d'ordre technique. Dans ce cas, le chinois a sacrifié la facilité d'apprentissage en faveur de la portabilité. Cela ne vous rappelle rien ?

En fait, le chinois n'est pas une langue unique. Il est constitué de quelque cinq langues principales [1], dont les formes parlées sont si différentes que leurs locuteurs ne se comprennent pas. On peut cependant écrire le chinois dans une langue et le lire dans une autre. Ça, c'est une langue vraiment portable. En choisissant un niveau plus élevé d'abstraction, le système d'écriture chinois optimise la communication plus que la simplicité. Un milliard de chinois ne peuvent pas tous se parler, mais au moins ils peuvent se passer des notes écrites.

Les ordinateurs aiment bien se passer des notes, eux aussi, grâce aux réseaux.

Une bonne partie de mes réflexions de cette année a été influencée par mon travail avec Unicode et XML. Il y a dix ans, Perl était très efficace pour traiter du texte. Il est encore meilleur maintenant, si l'on considère l'ancienne définition de « texte ». Mais cette dernière a changé sous ses pieds, pendant cette période.

La faute en revient à l'Internet.



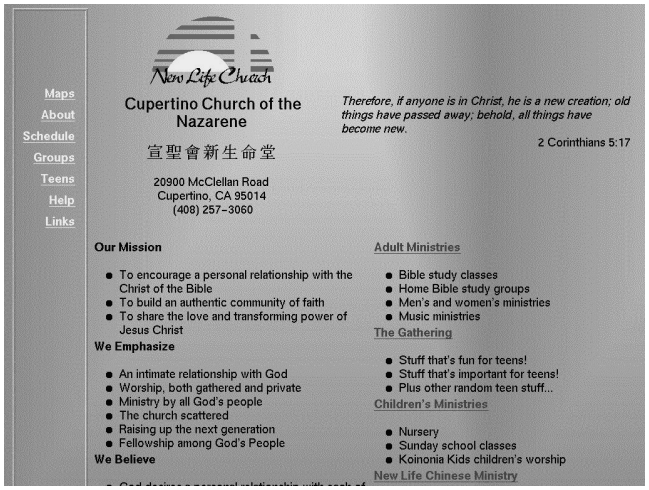
Il semble que, lorsque vous cliquez sur les boutons de votre navigateur, cela encourage des ordinateurs à échanger des notes. Et ceci, à travers toutes sortes de frontières culturelles. Tout comme vous voulez comprendre ce qui apparaît sur votre écran, votre ordinateur veut aussi comprendre ce qu'il va afficher à l'écran, parce que, que vous le croyiez ou non, l'ordinateur préférerait l'afficher correctement. Les ordinateurs sont peut-être stupides, mais ils sont toujours obéissants. Enfin presque toujours.

Et c'est là qu'Unicode et XML viennent s'insérer. Unicode n'est rien d'autre qu'un ensemble d'idéogrammes pour que tous les ordinateurs du monde puissent se passer des notes les uns aux autres, et aient une chance de faire ce qu'il faut avec. Il se trouve que certains des idéogrammes d'Unicode viennent coïncider

1. Pékinois, Mandarin...

avec des jeux de caractères nationaux comme l'ASCII, mais personne ne va apprendre toutes ces langues. Non seulement personne ne va apprendre toutes ces langues, mais nul ne l'attend de vous. Ce n'est pas le but.

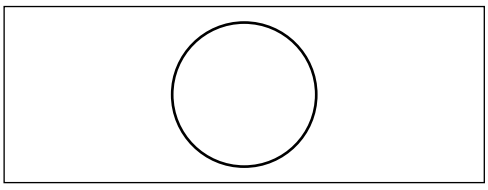
Voici le but. Le mois dernier j'étais en train de m'occuper de la page web de mon église qui vient juste de créer une congrégation chinoise. Elle a donc maintenant deux noms, dont l'un peut se représenter en ASCII, et l'autre non. Voici ce à quoi ressemble la page :



Si votre navigateur est assez récent, et si vous avez une fonte Unicode chargée, c'est ce que vous allez voir. Quelque chose d'important réside là.

Si j'avais réalisé ce document il y a un an, ce bloc de caractères chinois aurait probablement été une image au format GIF. Mais les images posent problème. On ne peut pas couper et coller des caractères à partir d'une image GIF. J'ai essayé assez souvent, et je suis certain que vous l'avez fait aussi. Et si j'avais voulu faire comme cela voici un an, j'aurais dû encore ajouter un niveau de complexité. J'aurais eu besoin de quelque chose comme un script CGI pour détecter si le navigateur comprend Unicode, faute de quoi ces caractères seraient apparus comme du bruit sur la page. Et on voit le plus souvent le bruit comme de la complexité inutile.

Revenons-en donc à la simplicité :

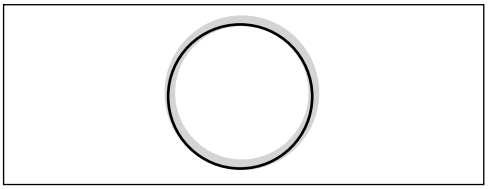


Nous utilisons les cercles pour représenter beaucoup de choses. Notre cercle d'amis. Une accolade, quand on le dessine sur le dos d'une enveloppe. Le cercle d'un anneau de mariage, qui représente l'amour éternel.

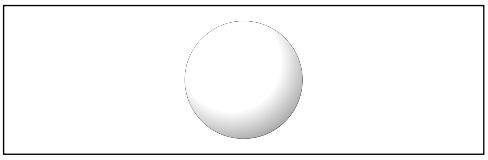
Pour passer du sublime au ridicule, nous avons aussi le classeur à anneaux, une sorte d'enfer pour de la paperasse inutile.

Des sphères lumineuses. Des trous noirs. Ou, au moins, leur horizon d'événements.

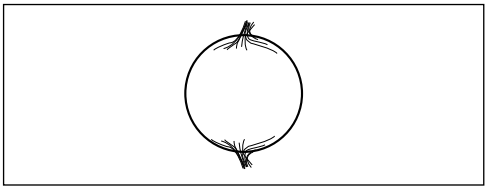
Un anneau pour tous les gouverner, et dans les ténèbres les lier.



Des boules de cristal. Des perles.

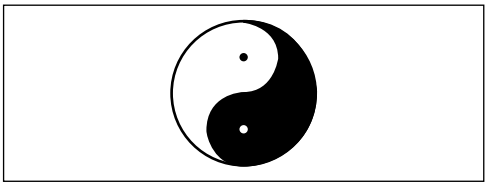


Des oignons. Des oignons-perles.



Les cercles figurent abondamment dans notre symbolisme. En particulier, en ajoutant diverses excroissances à des cercles, nous arrivons à représenter des notions assez compliquées avec des symboles simples. Ces symboles sont les ponts entre la simplicité et la complexité.

Voici un véritable diagramme Zen :

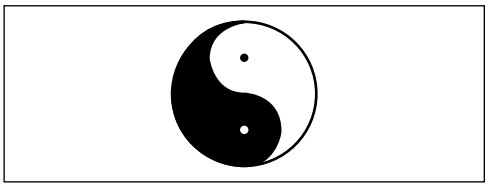


En fait, ce n'en est pas un ; le yinyang vient du Tao (ou Dao si vous ne pouvez pas prononcer un t non aspiré) qui est une très ancienne philosophie, plus vieille que le Zen d'au moins mille ans.

Revenons-en donc aux yins et yangs :

Le yinyang représente une philosophie dualiste, un peu comme La Force dans la guerre des étoiles. Vous savez en quoi La Force ressemble à un ruban adhésif ? La réponse est : parce que les deux ont un côté clair, un côté foncé, et ils unissent les différentes parties de l'univers. Personnellement, je ne suis pas dualiste, parce que je crois que la lumière est plus forte que l'obscurité. Cependant, l'idée de forces qui s'équilibrent est souvent utile, surtout aux ingénieurs. Quand l'un d'eux veut équilibrer de façon durable des forces, il va chercher son ruban adhésif.

Quand j'ai dessiné ce yinyang, je me suis demandé si je le faisais comme il faut. Ce serait dommage de le faire à l'envers, ou de travers, ou quelque chose comme ça.



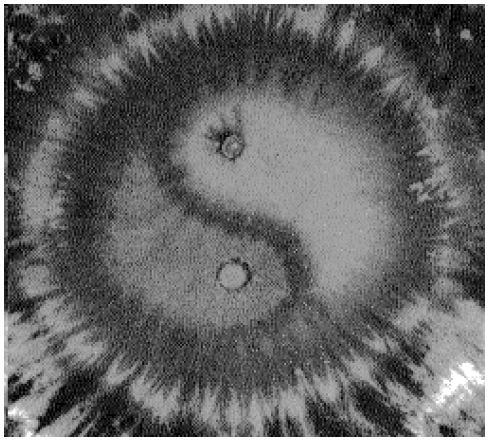
Vous savez, l'orientation est plus importante qu'on imagine, les spécialistes de chimie organique l'appellent « chiralité ». Si vous prenez une molécule d'essence de menthe et que vous la retournez, le résultat est une molécule d'essence de cumin. Beurk. J'étais convaincu que je détestais le pain de seigle, jusqu'au moment où je me suis rendu compte que c'étaient les graines de cumin que je n'aimais pas.

Préférer une de ces saveurs ou l'autre, ce n'est qu'une question de goût, mais les médecins et les spécialistes en chimie organique vous diront que la chiralité peut parfois être une question de vie ou de mort. Ou de membres déformés, dans le cas de la Thalidomide. Les dyslexiques vous diront que la chiralité est très importante dans le cas des symboles visuels. Ce texte contient les lettres b et d, et p et q, et le nombre 6, et aussi le nombre 9. Vous pouvez voir un 6 et un 9 dans le yinyang, dans cette orientation.

Je me demande en fait si le yinyang n'est pas un peu comme la swastika : le sens dans lequel vous les dessinez détermine qui va se fâcher avec vous.

J'ai donc un peu cherché, sur le web bien sûr. En fait, le web est un exemple parfait d'environnement où « il y a plus d'une façon de faire ». Dans ce cas, toutes les façons sont possibles. On peut trouver le yinyang dans toutes les orientations. Je ne sais toujours pas si l'une d'entre elles est plus correcte que les autres.

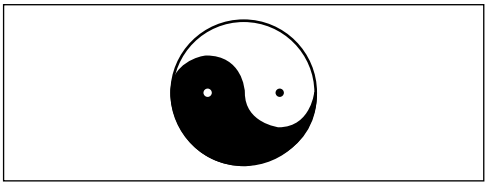
Les gens d'A TYEDYE WORLD vendent des t-shirts teints[2]. Pour eux le yinyang a cet aspect :



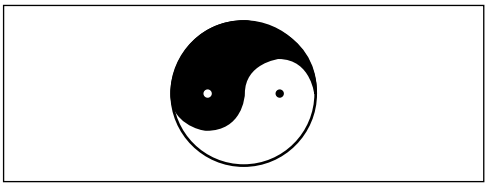
Je suppose que si vous le voulez dans l'autre sens, il suffit de retourner le t-shirt de façon à ce qu'on en voie l'intérieur. Vous pourriez aussi le mettre à l'envers, mais vous vous feriez un peu remarquer.

2. Tie-dyed c'est une sorte de batik. Les gens de A TYEDYE WORLD nouent des T-shirts, les teignent, les dénouent et les vendent (des Tao-shirts en ce qui nous concerne)

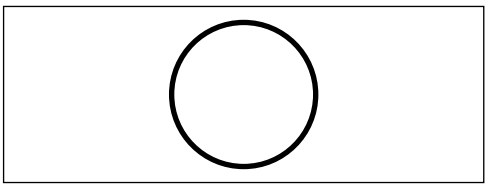
Les gens du consortium Unicode sont convaincus qu'il a cette forme. Je ne sais pas s'ils ont raison, mais si ce n'est pas le cas, ce n'est pas très important. Ils l'ont publié de cette façon, et maintenant c'est comme ça, par définition.



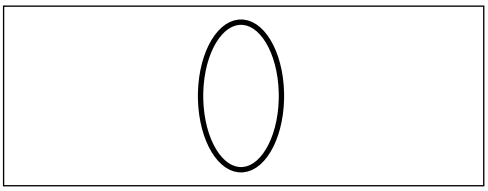
Bien sûr, mon dictionnaire le montre à l'envers :



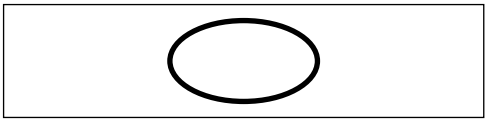
Revenons-en à Unicode. Unicode est plein de cercles. Beaucoup de jeux de caractères nationaux contenus dans Unicode utilisent le cercle, et pour nombre d'entre eux, il représente le chiffre 0. Voici le caractère Unicode numéro 3007 (hexadécimal). C'est le symbole idéographique de 0 :



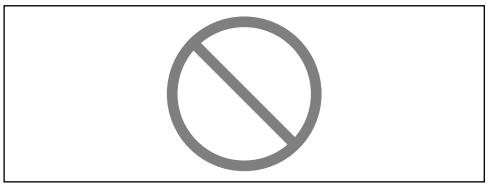
Surprise, il ressemble à notre 0. Un point pour l'impérialisme culturel. En anglais, bien sûr, nous avons tendance à rétrécir notre 0 sur les côtés, pour le distinguer de la lettre O.



En bengali, ils le rétrécissent dans l'autre sens, pour des raisons similaires :



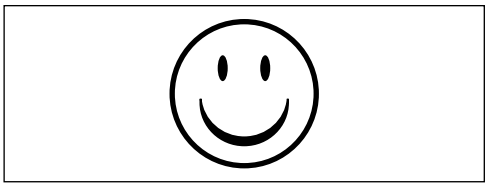
Je trouve intéressant que le monde contienne autant de représentations différentes pour rien. On pourrait en tirer bon nombre de blagues : « Beaucoup de bruit pour rien », ou « Rien ne peut arrêter une idée dont le moment est venu ». Voici quelque chose en rapport avec rien :



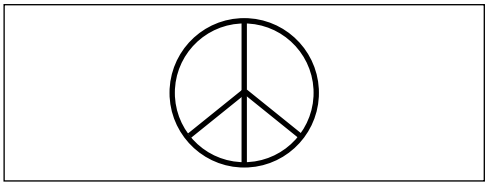
C'est le caractère universel pour « interdit ». En Unicode, il est classé comme un caractère composé.

Bien entendu, dans la culture Perl, presque rien n'est interdit. Je pense que le reste du monde contient déjà plein de bonnes interdictions, alors pourquoi en inventer de nouvelles ? Ceci s'applique à la programmation, mais aussi aux relations entre personnes. On m'a demandé plus d'une fois de bannir quelqu'un de la communauté Perl, parce qu'il avait été déplaisant d'une façon ou d'une autre. Jusqu'ici, j'ai toujours refusé, et je crois que j'ai eu raison. Du moins, au niveau pratique, cette façon de faire a fonctionné jusqu'ici. À chaque fois, soit l'individu en question a décidé de partir, soit il s'est calmé et a appris à s'arranger avec les autres de façon plus constructive. C'est étrange. Les gens comprennent instinctivement que la meilleure façon de faire des programmes qui communiquent entre eux est de les rendre stricts en ce qu'ils vont transmettre, et larges en ce qu'ils vont accepter. Ce qui est étrange, c'est que ces gens eux-mêmes ne sont pas prêts à être stricts en ce qu'ils disent et tolérants lorsqu'ils écoutent les autres. Cela semble pourtant évident, mais, à la place, on nous apprend à nous exprimer librement.

D'un autre côté, nous essayons d'encourager certaines vertus dans la communauté Perl. Comme l'apôtre Paul l'indique, personne ne fait de lois contre l'amour, la joie, la paix, la patience, la gentillesse, la bonté, la douceur, le contrôle de soi. Donc, plutôt que de nous efforcer à interdire les mauvaises choses, efforçons-nous de promouvoir les bonnes. Voici un caractère Unicode pour cela :

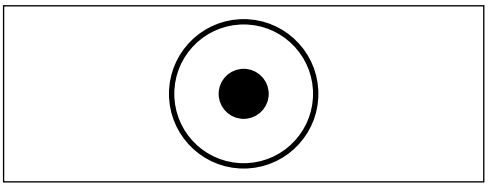


Bien sûr, si vous êtes un hippie, vous préférerez celui-ci :



Certains caractères Unicode à connotation positive sont moins évidents :

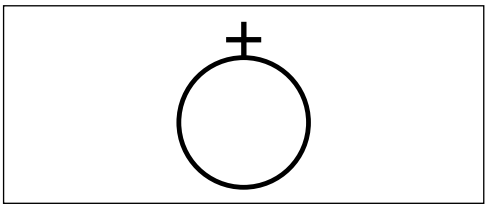
Voici le symbole pour le clic bilabial, un des symboles de l'Alphabet Phonétique International. Vous ne le connaissez peut-être pas, mais nombre d'entre vous font ce bruit régulièrement. Si vous voulez essayer, voici comment faire : vous joignez vos lèvres, et faites, avec la bouche, une espèce de son affriqué ingressif .



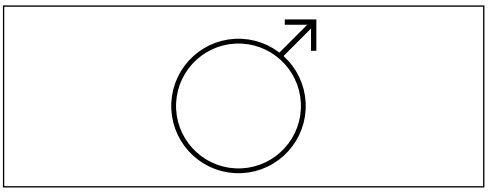
En anglais, nous notons ceci avec des X et des O au dos des enveloppes. Mais nous sommes témoins de la fin d'une époque. Avec l'arrivée en force du courrier électronique, l'habitude se perd d'envoyer des bisous et des accolades sur le dos des enveloppes. Et ça n'a pas vraiment le même effet dans une ligne d'en-tête de message électronique. Content-type : text/accoladebisous.

Vous voyez, c'est tout aussi difficile de parfumer un message électronique. Content-type : text/parfumé. De quoi exciter l'imagination.

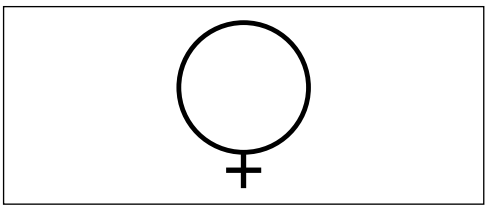
Voici encore des cercles qui représentent des choses compliquées. Celui-ci représente la Terre :



Et celui-là. Mars :



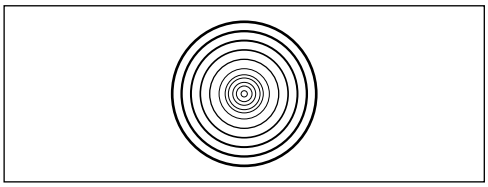
Et voici le symbole de Vénus :



À un certain moment j'ai travaillé pour le Jet Propulsion Laboratory, et j'ai un peu aidé à découvrir que Mars et Vénus sont très compliquées. Et, comme si les choses n'étaient pas assez compliquées comme ça, les anciens les compliquaient encore plus en réutilisant ces symboles pour représenter le mâle et la femelle. Pour eux les hommes venaient de Mars, et les femmes de Vénus. L'idée n'est pas nouvelle.

Voici encore un peu d'Histoire :

Si vous coupez un oignon, il ressemble à ça. Et si on le prend pour une métaphore du monde Perl, alors je dois être ce petit morceau d'oignon au milieu.

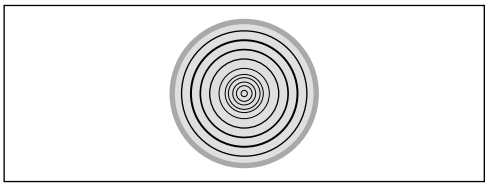


Autour de moi se trouvent certains des premiers adeptes de Perl, qui sont maintenant vénérés comme des héros de la révolution. Au fur et à mesure que le mouvement s'est élargi, de nouvelles couches ont été ajoutées. Vous pouvez aussi voir cela comme un atome, avec des couches d'électrons. Comme aucun atome connu ne contient autant de couches d'électrons, nous en resterons aux oignons.

Ce qu'il se passe avec l'oignon, c'est qu'il m'indique quelque chose sur ma propre importance, ou plutôt sur mon peu d'importance. C'est-à-dire que même si j'ai lancé tout ce mouvement, je ne suis qu'un petit morceau de l'oignon. Presque toute la masse se trouve sur les couches extérieures (c'est pourquoi j'aime voir apparaître des mouvements populaires, comme les Perl Mongers). Et je suis là, au milieu. On me reconnaît une importance historique, mais en fait les gens voient l'extérieur de l'oignon, pas l'intérieur. À moins qu'ils ne passent leur temps à découper des oignons en rondelles. Et même si c'est le cas, il y a plus d'oignon dans les grandes rondelles que dans les petites. Que ceux qui veulent être au centre en tirent une leçon. Ce n'est pas là que se trouve le véritable pouvoir. Pas dans ce mouvement, en tout cas. J'ai essayé de façonner ce mouvement selon un autre mouvement auquel j'appartiens, et le fondateur de ce mouvement disait : « Celui qui veut être le meilleur d'entre vous doit devenir le serviteur de tous ». De ceux qui étaient au centre avec lui, un l'a trahi, et les onze autres sont morts en martyrs. Mais je ne demande pas à mes amis de se donner en pâture aux lions, du moins pour l'instant.

Revenons-en aux motifs de croissance. Les perles naturelles croissent aussi par couches, autour d'un grain de sable qui irrite l'huître en question, qui forme de jolies couches autour. Ceci pourrait être une section d'une perle. C'est donc encore plus vrai avec les perles qu'avec les oignons : la couche externe est la plus importante. C'est la seule qui est visible. Ou, si la perle est en pleine croissance, c'est sur elle que va s'appuyer la couche suivante. Je vois bien que ceci me classe comme un simple irritant. Cette classification me va.

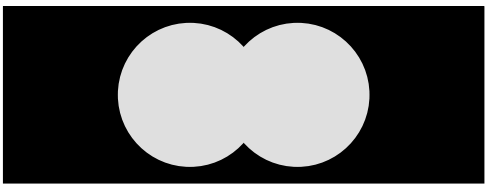
D'autres choses croissent avec le temps aussi. Ce sera peut-être plus clair si nous regardons les anneaux de croissance d'un arbre :



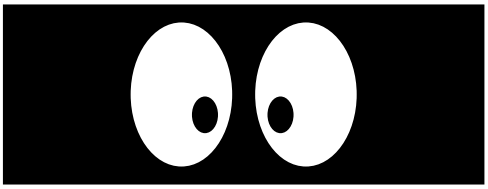
Si vous avez quelques notions de physique, vous saurez qu'un tube est presque aussi résistant qu'une barre solide du même diamètre, parce que la plus grande partie de la force se transmet par les couches externes. En fait, le centre d'un arbre peut être pourri, alors que l'arbre lui-même reste en bonne santé. Tous les ans, des gens économisent des milliards de dollars en programmant en Perl, mais

presque toutes ces économies se réalisent dans les tranchées les plus extérieures. Même plus près du centre, plus d'efforts visent à connecter Perl avec d'autres choses, qu'à modifier Perl lui-même. Et je suis d'accord avec cette façon de faire. Le centre de Perl est en train de se stabiliser. Même avec des changements centraux comme le support de multiples fils d'exécution (threads en anglais) ou d'Unicode, nous faisons comme si nous ajoutions des modules d'extension, parce que c'est plus propre, et que personne ne se trouve forcé à utiliser les nouvelles fonctions s'il ne le souhaite pas.

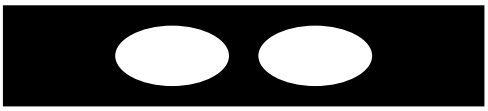
Toute cette histoire d'anneaux de croissance est très bien pour parler du passé, mais qu'en est-il du futur ? Je n'ai pas de boule de cristal, mais j'ai une paire de jumelles. En voici le symbole habituel :



Ceci est, bien sûr, le méthode habituelle pour indiquer, au cinéma, que quelqu'un est en train de regarder à l'aide de jumelles. Je ne sais pas trop quoi mettre dans le champ de vision ; voyons donc ce qui se trouve de l'autre côté des jumelles :



Ceci pourrait aussi être un dessin de deux corps célestes en rotation l'un autour de l'autre :



Chacune de ces planètes provoque des marées sur l'autre. Les gens comprennent en général pourquoi il y a un renflement sur le côté qui fait face à l'autre planète, mais plus difficilement pourquoi il y en a un également sur le côté opposé. Cela se comprend mieux si l'on considère que la deuxième planète, en plus de tirer sur le renflement le plus proche de la première, est aussi en train de tirer sur le centre de la première, l'éloignant du renflement opposé.

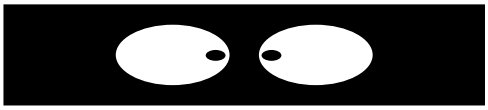
Voici une bonne image de la relation entre la communauté du logiciel libre, et celle du logiciel commercial. Nous pourrions même donner des noms aux extrémités. Inventons donc quelques noms. On pourrait appeler le bout à gauche, euh, Richard. Et on pourrait appeler le bout à droite, euh, Bill.

Les renflements au milieu sont plus difficiles à nommer mais, pour une fois, on pourrait appeler Larry celui du milieu à gauche, et Tim celui du milieu à droite.

Ceci constitue encore, bien entendu, un cas de simplification excessive, parce que les gens et les organisations ne se trouvent pas à un seul point donné,

mais ont tendance à bouger un peu. Quelques-uns réussissent à osciller entre un renflement et l'autre. À un moment ils sont pour une plus grande coopération entre les communautés du libre et du commercial, et juste après ils s'occupent à attaquer tout ce qui est commercial. Au moins nos hypothétiques Richard et Bill restent-ils cohérents.

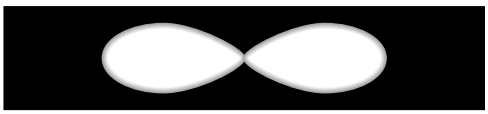
Mais l'action se déroule au milieu.



C'est le milieu que de plus en plus de personnes observent, pour voir ce qui va se passer. En fait, cette image correspond plutôt à l'année dernière. Cette année elle ressemble plus à ceci :

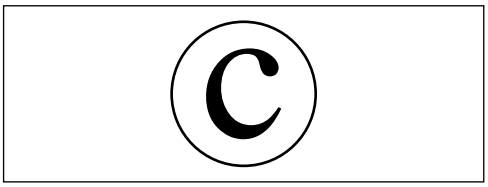
Robert L. Forward a écrit un roman, en fait une série de romans, sur un endroit nommé Rocheworld. Comme on pourrait s'y attendre, il tire son nom d'un individu nommé Roche. C'est lui qui a défini la limite de Roche, qui prévoit que des planètes se désagrégeraient si elles se rapprochaient en deçà de cette limite. Il s'est avéré qu'il avait trop simplifié, et que si vous laissez vos planètes prendre des formes comme celle-ci, vous pouvez les rapprocher beaucoup plus, sans compromettre leur stabilité. Bien sûr, la force gravitationnelle totale qui s'applique sur ces points est très faible, mais elle est suffisante pour assurer la cohésion des planètes.

De façon similaire, les communautés des logiciels libres et commerciaux sont beaucoup plus proches maintenant que beaucoup de gens ne l'avaient cru possible, avec leurs calculs. Dans le monde de Roche, les planètes ne se touchaient pas, mais elles avaient l'atmosphère en commun. Si nous rendons l'image un peu plus floue par la magie de xpaint, nous obtenons ce genre de résultat :



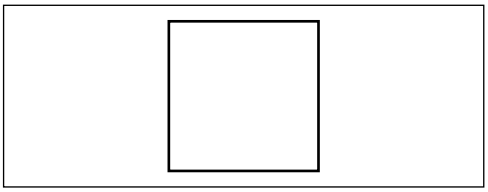
On voit ici comment on peut prendre l'avion d'une planète à l'autre, mais pas y aller à pied. Cela rappelle les tunnels quantiques, où il est impossible d'aller d'un endroit à un autre, sauf par un saut qui ne parcourt pas les points intermédiaires.

Un énorme flux d'idées circule entre les communautés du logiciel libre et commercial. Ensemble, ces deux lobes internes constituent ce que l'on appelle maintenant le mouvement Open Source. Et quelque chose de tout nouveau : des ennemis d'antan qui se mettent d'accord sur un bien commun au-delà de tout modèle commercial. Et ce bien commun, c'est plus de logiciel de qualité, et plus vite. Et cela a été possible, parce que les gens se sont rendus compte de la puissance d'une idée simple : on n'a pas besoin de brevets sur le logiciel, ni de secrets commerciaux. Il nous faut juste encore un cercle :



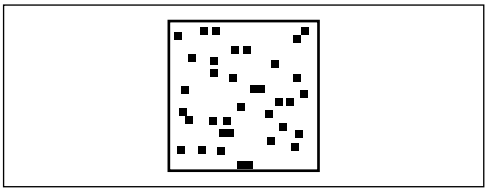
Un cercle avec un « c » à l'intérieur. L'Open Source devra sa survie ou sa mort aux lois sur le copyright. Nous espérons qu'il survivra. Faisons donc tous ce que nous pouvons, pour que ce soit possible. Si vous avez l'occasion d'argumenter pour le copyright plutôt que les brevets, faites-le. Respectons aussi la loi du copyright, et les souhaits des auteurs, qu'ils aient ou non été exprimés à la satisfaction des avocats de l'un ou l'autre. Le « c » dans le cercle devrait aussi représenter le civisme.

Et quand nous pensons au civisme, nous pensons aussi aux villes, et aux choses claires et carrées. Voici donc un carré :



Et, en effet, les villes sont faites de carrés, et de rectangles. Nous les appelons des blocs, ou des pâtés de maisons, ou des places quand les architectes urbains laissent un bloc sans bâtiments.

Parfois, les bâtiments eux-mêmes sont carrés :



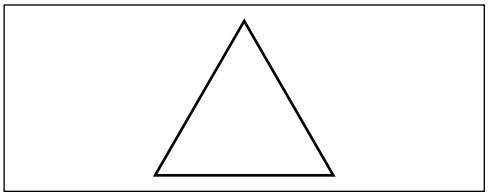
Mais, le plus souvent, ils ne le sont pas. De même, si vous regardez dans le livre d'Unicode, il y a nettement moins de carrés que de cercles. Je crois qu'il existe une raison fondamentale derrière cela. Quand nous construisons des bâtiments, et quand nous écrivons des caractères, nous les mettons dans une structure faite de lignes droites. En termes d'écriture, nous écrivons de gauche à droite, ou de droite à gauche, ou d'en haut en bas. Les cases abstraites, dans lesquelles nous mettons les caractères ou les bâtiments, sont plus ou moins carrées. Et les caractères, comme les bâtiments, ont tendance à disparaître visuellement, s'ils suivent les mêmes lignes que le texte lui-même. De nombreux caractères contiennent donc des lignes inclinées, tout comme l'on tente d'éviter que les gratte-ciels modernes ressemblent à des boîtes. Personne n'aime les gratte-ciels des années 60, parce qu'ils ressemblent trop à des boîtes. Les gens aiment que les choses se distinguent de leurs environnements.

C'est aussi pour cela que les différentes classes d'opérateurs et de variables en Perl se distinguent visuellement entre elles. Ce n'est que de la bonne ingénierie humaine, en ce qui me concerne. Je n'aime pas le fait que tous les opérateurs

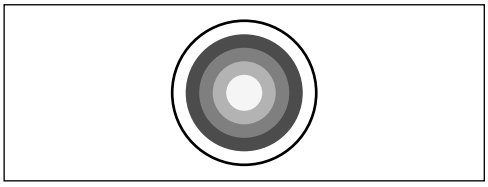
se ressemblent en Lisp. Je n'aime pas le fait que la plupart des plaques de rue se ressemblent en Europe. Et j'applaudis la décision de l'Allemagne, de faire des signes de stop différents de tous les autres. Bien sûr, c'est aussi une bonne chose pour nous américains ignorants qu'ils les ont fait comme des signes de stop américains. Un point de plus pour l'impérialisme culturel.

En repentance de l'impérialisme culturel américain, je vais indiquer encore un avantage des systèmes d'écriture idéographiques. Comme les idéogrammes s'écrivent dans des cases carrées, on peut tout aussi bien les écrire horizontalement que verticalement. Et vice-versa. Nos caractères à taille variable n'ont pas cette propriété. Surtout pas avec une police comme Helvetica, où l'on a du mal a distinguer les l des i même lorsqu'ils sont les uns à côté des autres. Si vous les alignez verticalement, le résultat sera juste une ligne cassée. Les chinois, les japonais et les coréens reviennent au score !

Pour conclure, je voudrais parler de triangles. En voici un :

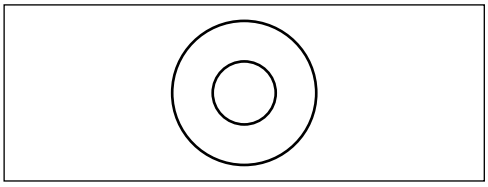


Les triangles sont aux cercles ce que les pointes de flèches sont aux cibles. Voici une cible :



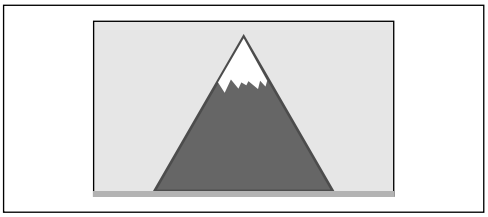
Je suis sûr de ne pas m'être trompé cette fois-ci. J'ai regardé sur le web. Et surtout, je me suis arrêté dès que j'en ai trouvé une.

En fait, ceci est le caractère Unicode nommé « œil de bœuf ».

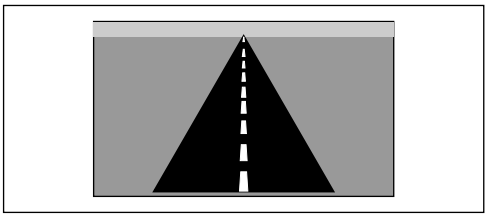


Je ne suis pas tout-à-fait sûr de ce qu'il est censé signifier, mais ce n'est pas un problème. Il suffit de lui inventer une signification.

J'ai tiré un bon nombre de flèches dans cet essai, et je ne sais pas si j'ai atteint des cibles ou pas. Nous mettons des triangles à la pointe des flèches parce qu'ils sont pointus. Nous associons les triangles à la douleur, surtout si nous leur marchons dessus. Les angles du triangle suggèrent la difficulté d'escalader une montagne :

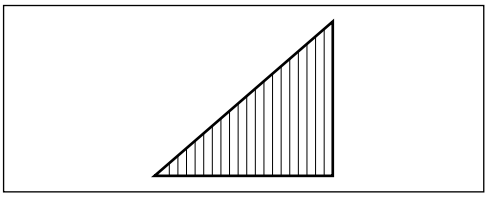


En même temps, les impressions peuvent être trompeuses. Un triangle représente aussi une route plate qui s'éloigne vers l'horizon :



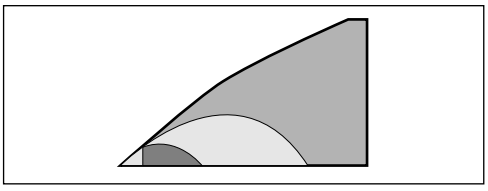
Ce n'est qu'une question de perspective. Vous pouvez choisir votre champ de vision en choisissant où vous placer. Je ne peux pas prédire si le chemin que prendra Perl sera plein de bosses ou pas, mais je peux prédire que, plus nous arrivons à voir de perspectives, mieux nous arriverons à choisir les perspectives que nous préférons. Et c'est, après tout, le travail du créateur d'un langage : étudier le problème depuis de multiples perspectives, être juste un peu omniscient, pour que le plus de monde possible en tire profit. Je fais un peu de triangulation, et j'organise le territoire. C'est mon travail. Si ma carte vous mène là où vous souhaitez vous rendre j'en serai heureux.

Si vous prenez une section de l'oignon qu'est Perl, il ressemble un peu à un triangle. Mettez-le sur le côté, et vous avez un graphique de la croissance de Perl pendant les dix dernières années.



Et c'est très bien comme ça. Ce graphique n'est que virtuel, bien sûr. Je n'ai aucune façon de mesurer la véritable croissance de Perl. Mais il est évidemment encore en train de croître. Nous faisons beaucoup de choses comme il faut, et en grande partie, il faut juste que nous continuions à faire ce que nous faisons.

Supposons maintenant que l'on rétrécisse ce triangle, et qu'on étende le graphique pour montrer toute la durée de vie de Perl. Nous n'avons vraiment aucune idée de ce que peut être cette durée.



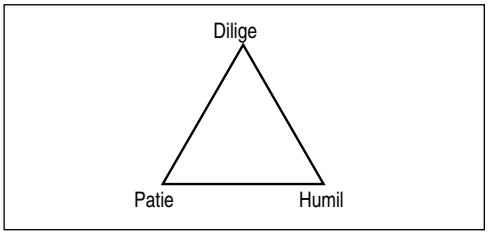
Il est difficile de prévoir quels vont être les facteurs décisifs pour cela. Mais je dois vous dire que je n'évalue pas le succès de Perl en en fonction du nombre de personnes qui m'apprécient. Quand je trace ces courbes, je compte les personnes ainsi aidées.

Je peux vous dire que je crois que la différence entre les courbes 1 et 2 peut dépendre de l'addition de tous les utilisateurs potentiels de MS-Windows, et tous les problèmes qu'ils doivent résoudre, qui sont nombreux. Ce n'est pas par hasard que nous venons de publier une boîte à outils Perl pour Win32.

Et je peux aussi vous dire que la différence entre les courbes 1 et 3 peut dépendre de l'addition de tous les utilisateurs internationaux qui pourraient bénéficier de Perl. Ce n'est pas par hasard que la dernière version en développement de Perl vous permet de nommer vos variables en utilisant n'importe quels caractères alphanumériques en Unicode, dont les idéogrammes. Il y a un milliard de gens en Chine. Et je veux qu'ils puissent se passer des notes écrites en Perl, et aussi écrire des poèmes en Perl.

Telle est ma vision du futur, la perspective que je choisis.

J'ai commencé en parlant des vertus du programmeur : la paresse, l'impatience, et l'orgueil.



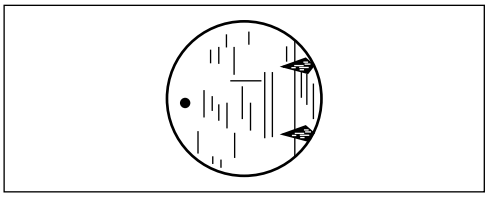
Ce sont des vertus de passion, et aussi des vertus individuelles. Ce ne sont pas, en revanche, des vertus pour une communauté. Les vertus d'une communauté sont opposées : la diligence, la patience, et l'humilité.

Elles ne sont pas vraiment opposées, parce vous pouvez toutes les pratiquer en même temps. C'est une question de perspective. Ce sont ces vertus qui nous menèrent jusqu'ici. Et ce sont ces vertus qui vont amener notre communauté vers le futur, si nous ne les abandonnons pas.

Avant tout, il faut que nous restions sur notre chemin. C'est ce que Friedrich Nietzsche appelait une « longue obéissance dans une même direction », ce qui constitue un bon slogan. Mais j'aime la citation complète aussi :

« Le plus essentiel « sur terre comme aux cieux » est d'avoir une longue obéissance dans une même direction ; de là en sort, et en est toujours sorti, à long terme, quelque chose qui rend la vie bonne à vivre. »

Et là nous avons bouclé la boucle, pour en revenir au cercle. Voici la porte d'entrée de la maison de Bilbo Baggins (Bilbon Sacquet). Un chemin naît devant cette porte, et Bilbo a écrit un poème :



Le chemin continue et s'éloigne
de la porte d'où il est parti.
Bien loin d'ici il arrive à présent,
et je dois le suivre, si je le peux,
de mes pieds impatients,
jusqu'à rejoindre une voie plus large,
où confluent des routes et des courses.
Pour aller où ? Je ne saurais le dire.

J.R.R. Tolkien, *The Hobbit*

Chapitre 11

La stratégie commerciale fondée sur les logiciels Open Source

En 1997 et 1998, des logiciels libres tels que Linux, FreeBSD, Apache et Perl suscitèrent une attention croissante de la part d'un nouveau public composé de chefs de projet, de cadres, d'observateurs de tendance et d'investisseurs.

La plupart des développeurs de ces logiciels accueillirent favorablement cet intérêt pour leur travail : pas uniquement parce que cela développait leur ego mais aussi parce qu'il leur permettait de justifier auprès de leurs supérieurs et de leurs collègues des efforts de plus en plus liés à leurs occupations professionnelles.

Mais ce nouveau public posait des questions difficiles :

- L'approche Open Source est-elle vraiment une nouvelle façon de concevoir un logiciel ?
- Chacun des succès des logiciels libres relève-t-il d'un concours de circonstances ou bien existe-t-il une méthode sous-jacente ?
- Pourquoi diable voudrais-je octroyer des ressources financières au développement d'un projet dont mes concurrents pourraient librement utiliser le code source ?
- Quelle confiance accorder à ce modèle de développement au-delà du cas d'un hacker fou ou de l'étudiant en informatique qui vient juste de réussir à accorder deux octets, lorsque l'on souhaite obtenir un logiciel correct ?
- Cette méthode menace-t-elle ou rend-elle obsolètes celles de ma société quant à la mise au point des logiciels et à leur mode de commercialisation ?

Je vous soumets l'idée que le modèle Open Source est indéniablement un modèle fiable de conduite du développement d'un logiciel commercial. Je veux essayer de vous montrer les conditions pré-requises pour un tel projet, quel type de projet se rattache à ce type de modèle, et les étapes qu'une entreprise devra franchir pour le lancer. Cet essai est destiné aux entreprises qui vendent des logiciels ou de l'assistance technique, ou qui utilisent un logiciel bien spécifique comme cur de leur activité (idée de pièce maîtresse).

11.1 Tout cela dépend des plates-formes

Bien que fervent partisan de l'approche Open Source en ce qui concerne les développements de logiciels, je reconnais volontiers que cette approche ne profitera pas toujours aux parties en présence. Elle exige des efforts, et les retours sur investissement ne sont jamais garantis. Une analyse correcte passe par la détermination des objectifs à long terme de l'entreprise et de ses points forts du moment.

Commençons par un examen des « Application Programming Interfaces » (API) [1] des plates-formes et des conventions. Pour les besoins de cet essai, je m'attarderai sur les API (telles que celle du serveur Apache pour la mise au point de modules), les protocoles régissant l'activité en ligne (par exemple HTTP), et les « conventions » érigées par les développeurs de systèmes d'exploitation (telles que la façon dont Linux organise son système de fichiers, ou encore dont les serveurs NT sont administrés) et désignerai tout cela par le terme « plate-forme ».

Win32, la collection de fonctions fournies et définies par Microsoft pour tous les développeurs d'applications MS-Windows 95 et NT, est une plate-forme. Si vous désirez développer une application destinée à des utilisateurs de MS-Windows, vous devez utiliser cette API. Si vous essayez, comme IBM le fit une

1. fonctions du système utilisables par le développeur d'application

fois avec OS/2, d'écrire un système d'exploitation qui puisse faire fonctionner des programmes mis au point pour Microsoft Windows, vous devez intégrer l'API Win32 dans son intégralité, afin d'être en mesure de l'exploiter.

De la même manière la Common Gateway Interface (ou CGI) est une plate-forme. Les spécifications CGI permettent aux développeurs d'un serveur Web d'écrire des scripts et des programmes qui fonctionneront avec n'importe quel serveur Web. CGI est beaucoup plus simple que Win32 et bien sûr permet beaucoup moins, mais son existence était importante pour le marché des serveurs Web parce qu'il permet aux développeurs d'applications d'écrire un code portable, des programmes qui fonctionnent avec n'importe quel serveur Web. Une différence capitale entre CGI et Win32, outre leurs niveaux de complexité, est que nulle entité ne possède les spécifications de CGI, donc que tous les principaux serveurs Web la proposaient afin de pouvoir exécuter n'importe quel script CGI. C'est seulement après plusieurs années d'utilisation qu'il fut envisagé de définir la spécification CGI par un document d'Appel à commentaires (RFC) publié par l'Internet Engineering Task Force [2].

Une plate-forme est ce qui définit de façon précise un quelconque morceau de logiciel, que ce soit un navigateur comme Netscape ou bien un serveur Web comme Apache. Les plates-formes permettent de construire ou d'utiliser un morceau de logiciel par-dessus un autre, et elles ne sont pas seulement essentielles pour l'Internet, où des plates-formes usuelles comme HTTP et TCP/IP facilitent vraiment la progression très rapide de l'Internet, mais sont aussi en train de devenir de plus en plus déterminantes lors de la phase de conception d'un environnement informatique, dans le cas d'un serveur comme dans celui d'une machine réservée à l'utilisateur.

Dans le cadre du projet Apache, nous avons eu la chance de développer d'emblée une API interne qui nous permette de distinguer d'une part les fonctionnalités centrales du serveur (chargé de gérer les connexions TCP, la coordination des sous-processus, et la gestion des requêtes HTTP de base), et, d'autre part, presque toutes les fonctionnalités de haut niveau comme le login, les modules CGI, les requêtes SSI (serveur side include), la configuration des paramètres de sécurité, etc. La richesse de cette API nous a aussi permis de mettre en place d'autres fonctionnalités importantes telles que `mod_perl` (un module d'Apache qui transforme le serveur Web en interpréteur Perl) et le `mod_jserv` (qui met en place l'API pour applets Java) afin de séparer les groupes de développeurs actifs. Le noyau du groupe de développement ne redouta donc pas de devoir non seulement maintenir et améliorer le cur du serveur mais aussi construire un « monstre » destiné à servir et recueillir les fruits de ces efforts intensifs.

Il existe des entreprises construites sur le modèle des plates-formes logicielles propriétaires. Elles peuvent tarifer tout usage de cette plate-forme sur la base d'une installation classique du logiciel, sur celle d'un paiement par utilisation, ou peut-être encore selon un autre modèle. Un copyright protège certaines d'entre elles, d'autres sont rendues opaques par l'absence de description rédigée pour le client ; d'autres encore évoluent si vite, parfois pour des raisons autres que techniques, que ceux qui tentent de les développer ne parviennent pas à suivre la cadence et semblent « à la traîne » sur le plan technique même si le problème ne relève pas de la programmation.

Un tel modèle commercial, bénéfique même à court terme pour l'entreprise

2. lire à ce propos l'essai de Scott Bradner (Chapitre 4, page 48).

qui possède la plate-forme, agit contre les intérêts de toutes les autres sociétés et ralentit le progrès technique. Des concurrents demeureront incapables de profiter de leurs compétences ou tarifs très étudiés parce qu'ils n'ont pas accès à la plate-forme. D'un autre point de vue, le consommateur peut être dépendant d'une plate-forme et, lorsque les prix augmentent, se trouver forcé de décider entre payer un peu plus sur le court terme pour la plate-forme ou bien dépenser beaucoup d'argent afin d'adopter une plate-forme plus viable.

Les ordinateurs et l'automatisation sont devenus si interdépendants et si essentiels pour l'entreprise qu'un responsable sensé ne doit pas confier à un seul fournisseur l'essentiel de son environnement. La possibilité de choisir un intervenant ne signifie pas seulement bénéficier du libre arbitre si certaines options sont trop coûteuses, et le coût d'un changement de plate-forme revêt à ce titre une importance capitale. Il peut être réduit si le logiciel n'est pas lié à la plate-forme. Les consommateurs gagnent donc à toujours exiger que le logiciel déployé soit adossé à une plate-forme non propriétaire.

C'est difficile à concevoir parce que les courbes rendant compte de l'offre et de la demande n'ont de sens que tant que les produits à vendre ont un coût facilement évaluable. Par exemple lorsque la fabrication de dix exemplaires d'un produit est à peu près dix fois plus onéreuse que celle d'un seul. Personne n'aurait pu prévoir l'extraordinaire économie d'échelle dont profitent les éditeurs de logiciels commerciaux, l'absence de corrélation directe entre l'effort requis pour produire un logiciel et le nombre de personnes qui peuvent l'acheter et l'utiliser.

Une personne ou un groupe qui produit des logiciels libres et qui crée un protocole fondamental ou une API est, à long terme, plus important pour la santé de cette plate-forme que deux ou trois créations indépendantes mais non libres. Pourquoi cela ? Parce qu'un concurrent peut acheter le produit commercial afin de le retirer du marché, réduisant à néant l'indépendance conférée aux utilisateurs par l'ouverture du logiciel. Cela peut aussi servir en tant que cadre académique ou référentiel de comparaison des réalisations et des comportements.

Des organisations, par exemple l'IETF ou le W3C, constituent un forum de développement de standards ouverts, et uvrent de façon quasi parfaite. Elles publient des documents généralement appréciés exposant comment les composants de l'Internet doivent interagir mais ne peuvent garantir le succès à long terme d'un standard donné, ni donc de sa pénétration. Ces organisations ne disposent d'aucun moyen d'imposer à leurs membres de créer des logiciels adossés aux protocoles ainsi définis. Le seul recours, parfois, consiste à fonder un groupe de travail capable de démontrer la supériorité de l'approche préconisée.

Par exemple, en décembre 1996, AOL a modifié de façon apparemment négligeable son serveur mandataire (proxy), utilisé par ses clients afin d'accéder au Web. Cette « mise à jour » relevait d'une arrière-pensée politique : lorsqu'un utilisateur accédait à un site utilisant un serveur Apache 1.2, à l'époque tout récent et conforme aux spécifications du nouveau protocole HTTP/1.1, on lui souhaitait la bienvenue grâce à ce message précis :

> VERSION WEB NON SUPPORTÉE
>
> L'adresse Internet que vous avez demandée n'est pas accessible via AOL. Le site Web demandé est en cause, et non AOL. Le détenteur de ce site utilise un langage HTTP non pris en charge. Si ce message apparaît fréquemment, vous pouvez changer les préférences

> graphiques pour l'Internet en COMPRESSÉ dans le menu : PRÉFÉRENCES.

Le noyau des développeurs Apache, alarmé, s'est réuni pour analyser la situation. Ils posèrent une question à l'équipe des techniciens d'AOL qui répondit :

> Les nouveaux serveurs Web HTTP/1.1 répondent sous forme HTTP-/1.1 à des demandes HTTP/1.0 alors qu'ils ne devraient générer que des réponses HTTP/ 1.0. Nous voulons briser au plus tôt cette vague de comportements incorrects devenant un standard de fait. Nous espérons que les auteurs de ces serveurs Web modifieront leurs logiciels afin qu'il ne génèrent des réponses HTTP/1.1 qu'aux demandes de même nature.

Malheureusement, les ingénieurs d'AOL avaient l'impression erronée que les réponses HTTP/1.1 n'étaient pas compatibles avec les clients HTTP/1.0 ou les mandataires. Ils le sont pourtant. HTTP a été conçu de sorte que la compatibilité ascendante de toutes les révisions mineures d'une version (par exemple la version 1) reste assurée. Mais les spécifications pour HTTP/1.1 sont tellement complexes que sans une lecture complète et détaillée, on aurait pu croire que ce n'était pas le cas, et plus particulièrement avec les documents HTTP/1.1 disponibles à la fin de 1996.

Nous, développeurs d'Apache, devions donc décider soit de faire marche arrière (donner des réponses HTTP/1.0 à des demandes HTTP/1.0), soit de respecter les conventions. Roy Fielding, notre « gardien du HTTP », fut en mesure de nous montrer clairement combien le comportement du logiciel à ce moment était correct et bénéfique ; il y aurait des cas où des clients HTTP/1.0 voudraient migrer vers une conversation HTTP/1.1 en découvrant que ce serveur la prend en charge. Il était aussi important d'indiquer aux mandataires qu'une première requête 1.0 vers un serveur pouvait être suivie d'un échange en 1.1.

Nous décidâmes de camper sur nos positions et de demander à AOL de corriger son logiciel. Nous suspections que la réponse HTTP/1.1 qui posait actuellement un problème à leur logiciel était causée par leur négligence en matière de qualité de la programmation plutôt que par un défaut du protocole. L'expertise étayait notre décision. Apache gérait alors 40% des serveurs Web de l'Internet, et la version 1.2 était déjà fort répandue. Ils devaient donc décider de corriger leurs erreurs de programmation ou bien de déclarer à leurs utilisateurs qu'au moins 20% des sites Web de l'Internet leur seraient interdits. Le 26 décembre, nous avons publié une page Web décrivant la dispute, et diffusé l'information pour justifier notre action, non seulement à nos propres utilisateurs, mais aussi à plusieurs journaux importants tels que C|Net et Wired.

AOL décida de corriger son programme. À peu près au même moment, nous avons annoncé la disponibilité d'une modification du code source d'Apache destinée aux gestionnaires de sites soucieux de contourner le problème d'AOL jusqu'à ce qu'il soit rectifié, une correction qui obligeait le logiciel à fournir une réponse en HTTP/1.0 aux requêtes issues d'AOL. Nous étions résolus à ce qu'elle reste une correction « non officielle », sans suite ni suivi, et qu'elle ne deviendrait pas la configuration par défaut de la distribution officielle.

Plusieurs autres éditeurs de produits HTTP (dont Netscape et Microsoft) proposèrent des logiciels non compatibles avec Apache. Dans la plupart de ces cas, l'éditeur devait choisir entre résoudre ses bogues ou se passer des sites

qui deviendraient alors inaccessibles. Dans la plupart des cas il implémentait délibérément le protocole de façon incorrecte sur ses serveurs et ses clients. Son produit fonctionnait alors normalement en milieu homogène (serveurs et clients de même marque), mais au mieux imparfaitement avec un serveur ou un client d'un concurrent. Cela posait des problèmes plus délicats que celui d'AOL, car dans certains cas la majorité des utilisateurs ne rencontre pas de dysfonctionnement patent. Les conséquences à long terme de ces bogues (ou de bogues additionnels augmentant le problème) ne seront perceptibles que lorsqu'il sera trop tard.

Si aucun logiciel libre strictement conforme et très utilisé tel qu'Apache n'avait existé, les incompatibilités de ce genre auraient régulièrement augmenté et se seraient superposées, masquées par les mises en doute mutuelles et croisées ou des tours d'esprit Jedi (« Nous ne parvenons pas à reproduire le problème en laboratoire »), où la réponse à « J'ai un problème quand je me connecte avec un navigateur X à un serveur Y » est, « Eh bien, utilisez le client Y et tout fonctionnera bien ». À la fin de ce processus, on aurait obtenu au moins deux Webs mondiaux l'un construit avec le serveur Web X, l'autre avec Y, et chacun d'eux n'aurait servi que ses clients. Ce genre de sape du standard a connu plusieurs précédents historiques, tant cette politique (le « verrouillage ») se trouve ancrée dans la stratégie commerciale de nombreux éditeurs de logiciels.

Bien sûr, ceci aurait été désastreux pour tout le monde car cela aurait condamné tout utilisateur du protocole HTTP (les fournisseurs de contenu et de services, les développeurs de logiciels...) à maintenir deux serveurs distincts. Certains consommateurs firent peut-être pression afin de contraindre les fournisseurs à s'accorder sur le plan technique mais le marché invitait fermement ces derniers à « innover, différencier, mener l'industrie, définir la plate-forme », et cela aurait pu leur interdire de définir conjointement les protocoles utilisés.

Nous avons en fait constaté un désastre de ce type avec la partie client de JavaScript. Les différents navigateurs Web, y compris les différentes versions bêta d'un même navigateur, présentaient de telles différences de comportement que les développeurs durent créer du code capable de détecter les différentes révisions afin de sélectionner une forme adéquate ce qui augmenta les délais de réalisation des documents interactifs en JavaScript. Ce n'est que lorsque le W3C intervint et jeta les bases de DOM (Modèle Objet de Document) que l'on assista à une tentative sérieuse de créer un standard multipartite autour de JavaScript.

Certaines forces naturelles du monde économique actuel poussent à la déviation lorsqu'un logiciel propriétaire est adossé à une spécification. Elle peut naître de la plus petite incompréhension du document de spécification, et se résorbera d'autant moins facilement qu'elle sera corrigée tard.

J'affirme donc que construire ses services ou ses produits sur une plate-forme standard préserve la stabilité de votre activité économique. Le succès de l'Internet a montré non seulement combien les plates-formes communes facilitent la communication mais a aussi contraint les fournisseurs à penser davantage à comment ajouter de la valeur à l'information en transit plutôt qu'à vouloir tirer profit du réseau lui-même.

11.2 Analyser les objectifs servis par un projet à code ouvert

Une entreprise doit donc déterminer dans quelle mesure ses produits implémentent une nouvelle plate-forme, et jusqu'où son intérêt commercial découle de la survie de cette plate-forme propriétaire. Quelle part de votre gamme de produits et de services, et donc quelle fraction de vos revenus, se trouve au-dessus de cette plate-forme, et combien au-dessous ? C'est même probablement quelque chose que vous pouvez chiffrer.

Supposons que vous éditiez un logiciel serveur de bases de données compatible avec de nombreux systèmes d'exploitation. Vous vendez séparément des logiciels d'administration graphique, des outils de développement rapide, une bibliothèque de procédures de base, etc. Vous proposez des contrats annuels d'assistance. Une mise à jour implique un nouvel achat. Vous assurez aussi des formations. Et votre groupe de conseillers croît rapidement et déploie votre logiciel auprès des utilisateurs.

Votre balance de revenus ressemble par exemple à ceci :

- 40% : Vente du logiciel
- 15% : Prestations d'assistance technique
- 10% : Travail de conseil
- 10% : Outils de développement rapide
- 10% : Bibliothèque de procédures/applications par-dessus la BD
- 5% : Documentation/formation

À première vue vous ne pouvez envisager d'offrir votre logiciel serveur car sa vente engendre 40% de vos revenus. Avec de la chance vous réalisez des bénéfices, et si vous êtes encore plus chanceux votre marge de profit atteint 20%. Renoncer à 40% brise l'édifice.

Cette analyse tient tant que l'équation ne varie pas. Il est cependant probable qu'elle change si vous étudiez correctement la question. Les serveurs de bases de données ne relèvent pas du groupe des applications que les entreprises achètent dans les rayons d'un magasin, installent, puis oublient. Toutes les autres catégories de revenus demeurent donc bien réelles et nécessaires quel que soit le prix du logiciel. Renoncer à facturer le serveur rend possible d'augmenter les autres services, car le coût du logiciel ne grève plus le budget des clients lorsqu'ils se procurent le serveur.

En simplifiant à outrance, si les utilisateurs restent motivés pour acheter les prestations de conseil et d'assistance ainsi que les outils de développement et les bibliothèques, alors la gratuité ou le moindre coût de la base de données permet de l'utiliser sur deux fois plus de systèmes. Cela ajoute donc 20% au revenu total. Mieux : cette gratuité permettra à environ trois ou quatre fois plus de nouveaux utilisateurs de découvrir votre logiciel. Cela réduira la proportion assurée par les autres services (car certains se contenteront de la version gratuite et aussi parce que vos concurrents offriront des services pour votre produit) mais tant qu'elle ne diminue pas trop vous augmenterez votre revenu global.

De plus, le coût de développement de votre logiciel diminuera si une licence appropriée le protège, notamment parce que des clients motivés corrigeront des bogues. La version standard intégrera peu à peu les efforts de ceux des consommateurs qui contribueront au projet de développement. Bref : vos coûts de développement diminueront.

Dans le cas d'une gamme de produits semblable à celle de l'exemple choisi, la « libération » du logiciel principal ne favorisera guère les actions de vos concurrents destinées à mieux exploiter d'autres sources de revenus. D'ores et déjà des conseillers intègrent vos outils, des auteurs indépendants en parlent, et d'autres entreprises (encouragées par vous) proposent des bibliothèques de code. La disponibilité du code source aidera peu vos concurrents à proposer une assistance technique portant sur votre code mais, en tant que développeur originel, vous disposerez d'un avantage qu'ils devront surmonter.

Tout n'est pas bleu et rose, bien sûr. Certains coûts liés à ce processus resteront difficiles à lier directement à un revenu. Le coût de l'infrastructure destinée à supporter cet effort, par exemple, n'est pas hors de portée mais peut mettre à l'épreuve les administrateurs de systèmes et le personnel assurant l'assistance technique. Il faudra y ajouter le coût de mise en place et de maintenance d'un canal de communication entre vos développeurs et le monde extérieur, et celui du développement d'un code ainsi publié, car la mise au point d'une version publiable du source s'avère parfois onéreuse. Après tout ce travail, le marché décidera peut-être que cette « libération » importe peu. Je fournirai à ce propos quelques réponses dans le reste de cet essai.

11.3 Évaluer les besoins du marché pour votre projet

Une entreprise perçoit souvent le logiciel libre comme un moyen de sauver un projet, de gagner en notoriété, ou simplement de placer de façon élégante une catégorie de produits sur une voie de garage. Ce ne sont pas de bonnes raisons pour lancer un projet Open Source, et une entreprise déterminée à poursuivre ce modèle a besoin de connaître exactement ce dont le produit a besoin pour garantir le succès d'une stratégie fondée sur du code ouvert.

La première étape consiste en une étude de l'existant, donc de tous les concurrents commerciaux ou libres, même les plus modestes. Soyez très attentif afin de déterminer ce que votre produit offre en séparant les parties qui peuvent être potentiellement offertes avec un produit commercial, vendues, ou bien dont le code source peut être publié séparément. De même, n'excluez pas des combinaisons de logiciels libres et commerciaux qui offrent les mêmes services.

Revenons à l'exemple du vendeur de base de données ci-dessus. Son produit comporte trois parties : un serveur SQL, un gestionnaire de transactions et de sauvegardes, et une bibliothèque de développement. Il ne doit pas seulement comparer ses produits avec ceux des leaders comme Oracle et Sybase, et ceux des petits concurrents dynamiques comme Solid et Velocis, mais aussi avec les bases de données libres telles que MySQL et Postgres. Une analyse de ce genre montrera peut-être que le serveur SQL apporte seulement quelques fonctionnalités supplémentaires à MySQL, et dans un domaine qui n'a jamais été considéré comme étant décisif sur le plan commercial mais simplement nécessaire pour rester en phase avec les concurrents. Le gestionnaire de sauvegardes et de transactions n'a aucun concurrent libre, et la bibliothèque Perl DBI surpasse la bibliothèque de développement proposée mais n'a pas de véritable concurrent en Java ou en C.

Cette société pourrait envisager les stratégies suivantes :

- Constituer un nouveau produit en remplaçant le serveur SQL par MySQL, auquel elle ajoutera les fonctionnalités spécifiques de son serveur SQL ainsi que le gestionnaire de sauvegardes et de transactions. De plus elle vendra les bibliothèques Java et C, et fournira la bibliothèque libre en Perl et l'assistance technique correspondante. Cela permettra de bénéficier de l'engouement pour MySQL et de l'incroyable bibliothèque de plug-ins et de codes additionnels proposés par des utilisateurs de ce logiciel. Cela vous permettra de conserver la propriété de tout le code breveté ou brevetable, ou simplement du code qui confère un avantage compétitif. Présentez-vous en tant qu'entreprise capable de déployer MySQL même au sein des environnements les plus étendus.
- Offrir la « fonctionnalité spécifique » aux développeurs de MySQL afin qu'ils l'y intègrent, puis modifier le gestionnaire de sauvegardes et de transactions afin de prendre en charge un grand nombre de bases de données, et afficher une nette préférence pour MySQL. Ceci a un revenu potentiel plus faible, mais permet à votre entreprise de rester plus focalisée et d'atteindre un plus large groupe de consommateurs. Un tel produit peut aussi être plus facile à maintenir.
- Aller dans la direction opposée : commercialiser le serveur SQL et les bibliothèques mais libérer le code source du gestionnaire de sauvegardes et de transactions, en tant qu'utilitaire général destiné à un grand nombre de types de serveurs. Cela réduirait votre coût de développement pour ce composant, et offrirait un avantage marketing à votre serveur. Cela réduirait un peu l'avance que vos concurrents auraient gagné grâce à l'Open Source, même si cela réduirait aussi un peu la vôtre.
- Libérer la totalité du serveur SQL en tant que tel, séparément de MySQL, de Postgres ou de tout autre logiciel existant, et vendre des contrats d'assistance technique pour votre logiciel. Vendre l'outil de sauvegarde mais libérer le code source des bibliothèques afin d'encourager de nouveaux utilisateurs. Cette stratégie augmente le risque car un logiciel populaire comme MySQL ou Postgres existe depuis déjà longtemps et un développeur manifeste d'ordinaire une réticence au changement si le logiciel déployé fonctionne correctement. Pour faire cela, vous devez souligner les avantages patents de votre produit pour que les utilisateurs soient tentés de l'essayer : plus rapide, plus souple, plus facile à administrer ou à programmer, ou riche de nouvelles fonctionnalités. Vous devez aussi consacrer davantage de temps aux actions susceptibles d'attirer l'attention de la clientèle potentielle, et trouver un moyen d'éloigner les développeurs des produits concurrents.

Je ne proposerais pas la quatrième solution dans ce cas, car MySQL a une très nette avance ici, beaucoup d'ajouts, et de nombreux utilisateurs.

Un projet de logiciel libre décline parfois à cause d'un manque de dynamisme des membres du noyau de l'équipe de développement, ou parce que le logiciel devrait s'affranchir des limites de sa propre architecture afin de satisfaire de nouveaux besoins, ou bien parce que la demande dépend d'un contexte révolu. Quand cela arrive, les gens cherchent alors d'autres solutions et il devient donc possible d'introduire un autre produit qui attirera l'attention même s'il ne constitue pas une avance significative sur le statu quo.

La demande engendre de nouveaux projets de logiciels libres, son analyse est donc absolument nécessaire. Le développement d'Apache commença lorsqu'un

groupe de webmestres partagèrent les corrections apportées au logiciel serveur Web du NCSA, puis décidèrent qu'échanger des corrections comme autant de cartes à collectionner restait peu efficace et sujet aux erreurs. Ils décidèrent donc de proposer une distribution séparée du serveur du NCSA incluant leurs modifications. Aucun des acteurs principaux des débuts ne s'est engagé parce qu'il voulait vendre un serveur commercial, bien que ce soit certainement une raison valable de le faire.

Une analyse de la demande du marché pour un projet particulier de logiciel libre impose aussi de s'abonner aux listes de diffusion et les forums Usenet pertinents, de fouiller les archives, et d'interroger vos clients et leurs collègues. Alors seulement vous pouvez déterminer si certains souhaitent participer au projet.

Revenons aux débuts du serveur Apache. Ceux d'entre nous qui échangions des corrections, les envoyions aussi au NCSA dans l'espoir de les voir inclus dans la distribution, ou tout au moins validés afin de nous faciliter leur intégration à des versions ultérieures. Puis Netscape embaucha certains des développeurs de ce serveur, et les autres ne parvinrent plus à gérer les messages électroniques expédiés par toutes les parties intéressées. Construire notre propre serveur fut plus une action d'auto-préservation qu'une tentative visant à proposer un serveur Web. Il est important de commencer avec des objectifs limités donc faciles à atteindre, et de ne pas devoir attendre que votre projet domine le marché avant de réaliser des bénéfices.

11.4 Position des logiciels libres dans la gamme de logiciels

Pour déterminer quels composants d'un produit donné libérer, il peut être intéressant de conduire un exercice simple. Premièrement, dessinez une ligne représentant une gamme de produits. Sur la gauche, inscrivez le libellé « Infrastructure ». Cette partie comprendra les logiciels qui implémentent les systèmes et plates-formes, jusqu'au noyau, à TCP/IP, et même jusqu'au matériel. Sur la droite, inscrivez « Applications », représentant les outils et les applications destinés à l'utilisateur non technicien. Le long de cette ligne placez des points représentatifs, en termes relatifs, où vous pensez que se situe chacun des composants de vos logiciels. Dans l'exemple ci-dessus, les outils d'administration et les frontaux graphiques se trouvent à l'extrême droite, tandis que le code qui gère les sauvegardes se trouve à l'extrême gauche. Les bibliothèques de développement se trouve plutôt à proximité du centre, et le moteur SQL à gauche. Ensuite, vous pouvez placer les produits de vos concurrents, en les détaillant par composant et en utilisant un stylo de couleur différente pour distinguer les logiciels libres des produits commerciaux. Vous constaterez vraisemblablement ainsi, que les logiciels libres sont souvent placés à gauche et les offres commerciales à droite du graphique.

Les logiciels libres sont donc plus souvent proches de l'infrastructure et des systèmes d'arrière-plan que de la gamme de logiciels représentée ici. Il y a plusieurs raisons à cela :

– Les applications finales sont difficiles à écrire, non seulement parce qu'un programmeur doit réaliser un environnement graphique fenêtré qui change

constamment, non standard, et bogué (ne serait-ce qu'à cause de sa complexité), mais aussi parce que la plupart des programmeurs ne sont pas de bons concepteurs d'interfaces, malgré de notables exceptions,
- Culturellement, le logiciel libre existe depuis longtemps dans le code réseau et le système d'exploitation,
- Le logiciel libre a tendance à prospérer là où les changements incrémentaux règnent et, historiquement, cela concerne les systèmes d'arrière-plan plutôt que les frontaux,
- Le plus gros du logiciel libre a été écrit par des ingénieurs pour résoudre une tâche assignée lors du développement de logiciels ou de services commerciaux. Le premier public était donc, dès le début, d'autres ingénieurs.

Voilà pourquoi nous constatons la présence de nombreux logiciels libres fiables pour les systèmes d'exploitation et les services réseaux, mais très peu pour les applications destinées à l'utilisateur.

Des contre-exemples existent. Par exemple GIMP, programme de manipulation d'images GNU, un programme X11 comparable, dans ses fonctionnalités, au « Photoshop » d'Adobe. Déjà, par certains aspects, ce produit est aussi un outil d'infrastructure, une plate-forme, car il doit son succès à sa merveilleuse architecture modulaire et aux douzaines de compléments développés pour permettre d'importer et d'exporter de nombreux formats de fichiers différents et d'employer des centaines d'effets de filtres.

Examinez à nouveau le graphique dessiné. Vous pouvez voir votre offre dans le contexte de vos concurrents, et dessiner une ligne verticale. Elle marque la séparation entre le code que vous libérerez et le reste. Cette ligne représente votre vraie plate-forme, votre interface entre le code public que vous essayez d'imposer comme standard sur la gauche, et le code propriétaire pour lequel vous souhaitez susciter de la demande sur la droite.

11.5 La nature a horreur du vide

Tout espace occupé par des logiciels commerciaux dans une infrastructure par ailleurs gagnée par le logiciel libre offre une forte motivation de redéveloppement dans l'espace public. Lorsqu'un mur commercial existe entre deux pièces de logiciels libres une sorte de force naturelle s'exerce afin de combler ce manque avec une solution publique parce qu'il peut être rempli avec des ressources suffisantes. S'il est suffisamment limité pour être traversé par votre entreprise avec votre propre équipe de développement, il est probable qu'il le soit suffisamment pour être comblé par un groupe de développeurs motivés.

En ce qui concerne notre exemple de la base de données, mettons que vous décidiez de publier les sources de votre serveur SQL (ou de votre propre amélioration ajoutée à MySQL) mais que vous commercialisiez un pilote propriétaire assurant une interface avec un serveur Web permettant de créer des contenus dynamiques. Le serveur de données sera un point d'attraction favorisant ce produit, et vous augmenterez donc substantiellement les marges de ce composant.

Relier une base de données à un serveur Web reste à la fois courant et recherché, les développeurs devront donc acquérir votre interface ou bien trouver un autre moyen d'accéder aux données depuis le serveur Web. Chaque développeur souhaitera économiser l'argent qu'il devrait vous verser. Si un nombre de développeurs suffisant mettent en commun leurs ressources, ou si une seule

personne compétente ne peut simplement pas payer pour votre pilote mais veut quand même utiliser la base de données, il est possible que vous trouviez un beau matin un concurrent libre à votre offre commerciale, ce qui annihile votre avantage.

Cet exemple reflète un concept plus large : s'appuyer sur un code source propriétaire dans une zone stratégique pour gagner de l'argent est devenu une aventure commerciale risquée. Si vous pouvez gagner de l'argent en assurant la l'assemblage du serveur Web, les interfaces et le serveur de données, ou en fournissant une interface qui gère ce système dans son ensemble, vous vous épargnerez ce type de surprise.

Tous les logiciels commerciaux ne présentent pas cette vulnérabilité. C'est même une caractéristique spécifique aux logiciels commerciaux qui essayent de s'installer dans une niche directement entre deux logiciels libres bien établis. Présenter votre offre commerciale comme une addition aux logiciels libres est une stratégie bien plus solide.

11.6 Dois-je offrir, ou bien me débrouiller seul ?

Beaucoup de catégories de logiciels abritent des implémentations libres, surtout sur le marché des serveurs. Par exemple des systèmes d'exploitation, des serveurs Web, des serveurs de messagerie (SMTP, POP, IMAP), de news (NNTP), et de noms (DNS), des langages de programmation (la « glu » pour un contenu dynamique sur le Web), des bases de données, du code réseau en tout genre. Sur votre bureau on trouvera des éditeurs de texte comme Emacs, Nedit et Jove, des systèmes graphiques tels que Gnome et KDE, des navigateurs Web tel que Mozilla ainsi que des économiseurs d'écrans, des calculatrices, des organiseurs, des clients de messagerie, des outils pour le graphisme La liste est longue. Bien que toutes les catégories ne recèlent pas de logiciels libres incontournables tels qu'Apache ou Bind, il y a très peu de niches commerciales sans au moins un concurrent libre en chantier. C'est beaucoup moins vrai pour la plate-forme Win32 que pour Unix ou Mac, principalement parce que la communauté ne la juge pas suffisamment libre.

Si un logiciel libre rencontre le succès dans une catégorie qui recoupe votre offre potentielle ne négligez pas que participer à son développement en y ajoutant de nouveaux modules permettra d'améliorer aussi le source de votre produit, de prendre un avantage sur le plan marketing, ou d'établir une plate-forme commune. Afin de déterminer si cette stratégie est acceptable on doit étudier sa licence :

- Les termes sont-ils en accord avec vos objectifs à long terme ?
- Pouvez-vous légalement y ajouter du code sous cette licence ?
- Encourage-t-elle suffisamment les développeurs ? Sont-ils prêts, sinon, à vous ménager une place en modifiant la licence ?
- Votre contribution est-elle assez générale pour apporter une valeur ajoutée aux développeurs et aux utilisateurs du projet existant ? Si elle ne constitue que l'implémentation d'une interface vers votre code propriétaire, elle ne sera probablement pas acceptée.
- Si votre contribution est importante, pouvez-vous obtenir un statut équivalent à celui des autres développeurs, pour pouvoir effectuer ensuite directement des corrections de bogues et des améliorations ?

– Pouvez-vous collaborer avec les autres développeurs ?
– Vos développeurs peuvent-ils collaborer avec d'autres ?

Satisfaire les développeurs est probablement le plus grand défi pour le modèle de développement du logiciel libre, on ne peut pas vraiment y répondre avec un amoncellement de techniques ni même avec de l'argent. Chaque développeur doit réaliser qu'il contribue au projet, que vous ne négligez pas ses questions et ses commentaires sur l'architecture et la conception du projet, et que le fruit de ses efforts sera intégré dans la distribution ou qu'il recevra une explication détaillée des raisons pour lesquelles il ne peut l'être.

Beaucoup se trompent car déclarent : « Le modèle de développement des logiciels libres fonctionne car, grâce à lui, l'Internet tout entier devient votre service de Recherche & Développement et de gestion de la Qualité ! ». En pratique le nombre total de développeurs disponibles capables de mener à bien une mission donnée reste limité, mieux vaut donc éviter qu'un projet parallèle naisse à cause d'une dispute d'importance secondaire séparant les développeurs. Mais la sélection naturelle fonctionne au mieux lorsque les projets luttent afin de profiter des ressources. Maintenir deux projets en compétition dans la même niche lorsque les développeurs ne manquent pas permettra à l'une des parties d'innover d'une façon totalement nouvelle pour les autres.

Le domaine du serveur SMTP offre un bel exemple de compétition salutaire. Le programme libre Sendmail d'Eric Allman régna longtemps sans partage, puis des concurrents apparurent, par exemple Smail et Zmailer. Mais Qmail, de Dan Bernstein, convainquit le premier de nombreux utilisateurs. Sendmail avait à ce moment 20 ans et commençait à accuser le poids des années. Il n'est pas conçu pour l'Internet de la fin des années 90, où lesăattaques par dépassement de la capacité des tampons (buffer overflow) et refus de service (denial of service) sont aussi fréquents que la pluie en Bretagne. Qmail fut à plus d'un titre une coupure radicale. La conception du programme, son administration, et même son mode opératoire étaient nouveaux. Sendmail ne peut vraisemblablement pas le rejoindre. Non parce qu'Allman et son équipe ne sont pas de bons programmeurs ou ne reçoivent pas d'aide efficace et motivée mais simplement parce que seul un changement radical de Sendmail leur permettrait d'explorer de nouvelles voies. Pour des raisons similaires IBM finance le développement du serveur SMTP SecureMailer de Wietse Venema, qui semble aussi devenir populaire en ce moment. Le domaine du serveur SMTP est suffisamment défini et important pour pouvoir soutenir plusieurs projets de logiciels libres. L'avenir dira lesquels survivront.

11.7 Amorçage

Il est essentiel pour la santé d'un projet à code ouvert de profiter d'un élan grâce auquel il d'évoluera, et de répondre à de nouveaux défis. Le monde du logiciel ne connaît rien de statique, et chaque composant majeur restera maintenu et amélioré. L'un des avantages de ce modèle est qu'il réduit la quantité de développement attribuée à chaque groupe. Vous avez donc besoin d'autres développeurs actifs.

Durant la phase de détermination de la demande vous rencontrerez probablement un groupe d'autres entreprises et de personnes suffisamment intéressés pour former un noyau de développeurs. Exposez-leur les grandes lignes de votre stratégie, sans la fixer. Il peut être utile de créer une liste de diffusion pour

discuter de ce projet avant de fixer quoi que ce soit. Ce groupe émettra probablement des idées pertinentes et dressera une liste de ses propres ressources mobilisables.

Dans le cas d'un projet peu ambitieux les volontaires pourront ne s'engager qu'à essayer votre produit et rester abonnés à la liste de diffusion. Il faudra, sinon, quantifier les ressources disponibles.

Voici les ressources minimum pour un projet de complexité modérée, comme par exemple le développement d'un outil générique de gestion d'un panier d'achats sur le Web, ou d'un nouveau type de serveur réseau implémentant un protocole simple. Je décrirai ci-après les rôles et types de compétences.

Rôle 1 : Coordination de l'infrastructure

Une personne installera et maintiendra la liste de diffusion et ses listes d'abonnés, le serveur Web, le serveur CVS (Concurrent Versioning System), la base de données des bogues, etc.

- Mise en place : 100 heures
- Maintenance : 20 heures/semaine

Rôle 2 : Coordinateur du code

Quelqu'un chargé de la coordination des engagements et responsable de la qualité du code implémenté. Il gérera aussi les corrections apportées par des tiers et résoudra les incompatibilités entre ces contributions et les bogues que leur intégration rend patents. Cela s'ajoute à tout développement dont il peut être chargé.

- Mise en place : 40-200 heures (suivant le temps nécessaire pour nettoyer le code avant qu'il soit rendu public).
- Maintenance : 20 heures/semaine

Rôle 3 : Maintenance de la base de données des bogues

Bien qu'il ne s'agisse pas ici d'assurer de l'assistance technique non rémunérée, il reste important que le public dispose d'un moyen sûr d'expédier aux développeurs leurs questions et les rapports de bogues. Au sein d'une organisation libre les développeurs ne sont, bien entendu, pas tenus de répondre à tous les messages reçus mais doivent déployer des efforts raisonnables afin de répondre aux remarques pertinentes. La maintenance de la base de données des bogues sera la première ligne de soutien, quelqu'un étudiera donc régulièrement les rapports et supprimera les questions auxquelles la documentation répond et les stupidités puis transmettra aux développeurs les descriptions de problèmes réels.

- Mise en place : juste assez pour apprendre à connaître le code.
- Maintenance : 10-15 heures/semaine

Rôle 4 : Maintenance de la documentation et du site Web

Les responsables de projets de logiciels libres négligent souvent ce poste et l'abandonnent donc parfois aux ingénieurs ou à des personnes souhaitant réellement contribuer mais n'ayant pas les connaissances techniques nécessaires. Elle

reste même souvent simplement vacante. En admettant que nous ne cultivions pas le secret il faut admettre qu'un logiciel doit, afin de gagner sans cesse de nouveaux utilisateurs, bénéficier des efforts d'intervenants capables de publier de l'information destinée aux utilisateurs potentiels. Cela aide à réduire le nombre de rapports de bogues redondants car procédant de déficiences de la documentation, et encourage aussi de nouvelles personnes à connaître le code puis à y contribuer à leur tour. L'importance d'un document décrivant l'architecture interne du logiciel est capitale, et celle d'un exposé des procédures ou classes principales du code l'est presqu'autant.

- Mise en place : 60 heures (en présumant que peu de code ait été documenté).
- Maintenance : 10 heures/semaine

Rôle 5 : Stratège/évangéliste/commercial

Une personne capable de susciter un élan pour le projet en trouvant d'autres développeurs, demandant à des utilisateurs potentiels spécifiques d'essayer le logiciel, trouvant d'autres entreprises candidates pour adopter cette nouvelle plate-forme, etc. Il ne s'agit pas ici seulement de marketing ou de prospection commerciale, mais d'une mission d'ordre technique dont le responsable présentera sa capacité à percevoir clairement le rôle du projet dans une perspective d'ensemble.

- Mise en place : assez pour connaître le projet
- Maintenance : 20 heures/semaine

Ces cinq rôles occupent presque trois personnes à plein temps. En réalité un groupe de personnes partageant les responsabilités pourront partager les responsabilités afférentes, et certains projets survivront, sitôt les premiers obstacles de publication franchis, alors que le niveau d'implication moyen des participants ne dépasse pas cinq heures hebdomadaires. Mais au début du projet les développeurs doivent concentrer leurs efforts comme s'il s'agissait d'un classique projet industriel d'entreprise.

Ces cinq rôles n'assureront aucun nouveau développement, tout cela ne relève que de la maintenance. Si vous ne trouvez pas assez de ressources auprès de collègues et de partenaires pour assurer tout cela ainsi qu'un nombre suffisant de nouveaux développeurs chargés du travail de base (jusqu'à ce que de nouvelles recrues se présentent) vous devriez reconsidérer la publication du code de votre projet.

11.8 Quelle licence adopter ?

Déterminer quelle licence utiliser pour votre projet s'avérera parfois difficile. Vous n'apprécierez probablement pas cette étude, mais votre service juridique s'en chargera. D'autres documents et sites Web traitent en détail de ces questions liées à la propriété industrielle. Je vais me contenter de proposer une vue générale sur la portée commerciale de chaque style de licence.

11.8.1 La licence de style BSD

Apache et les projets de systèmes d'exploitation issus de BSD (FreeBSD, OpenBSD, NetBSD) l'emploient, et il peut être résumé par : « Voilà le code, employez-le comme bon vous semble, cela m'est égal. Mais faites toujours état de sa provenance. ». D'ordinaire, cette reconnaissance de l'origine est requise sous différentes formes (publicité, fichier LISEZ-MOI, ou dans la documentation imprimée, etc). D'aucuns affirment que cette exigence rend l'approche incompatible avec toute mise à l'échelle, c'est-à-dire que si quelqu'un publie un ensemble comprenant 40 modules libres sous licence BSD différents il pourrait y avoir 40 notes de copyright à afficher. En pratique cela ne pose jamais de problème, et en fait certains perçoivent ces exigences comme autant de forces positives destinées à faire prendre conscience de l'importance des logiciels libres.

Dans une perspective commerciale la licence BSD offre le meilleur contexte légal à tout nouveau participant, car elle ne laisse peser aucune inquiétude quant aux restrictions d'utilisation ou de redistribution. Vous pouvez mélanger ou comparer ce logiciel avec votre propre code propriétaire, et ne publier que ce qui, selon vous, fait progresser le projet et vous aide en retour. C'est pourquoi nous l'avons choisie pour Apache. Notre logiciel a été lancé par des webmestres commerciaux en quête d'un meilleur serveur Web pour leurs propres besoins commerciaux, contexte fort différent de celui que connurent beaucoup d'autres projets libres en phase de gestation. Vraisemblablement aucun des membres de l'équipe d'origine ne se proposait de créer un serveur commercial, aucun de nous ne savait ce que le futur nous apporterait, et nous ne souhaitions pas limiter de prime abord nos possibilités de choix.

Le groupe des développeurs d'Apache a aussi adopté ce type de licence car elle hisse à merveille un code donné en tant qu'implémentation de référence d'un protocole ou d'un service commun. Beaucoup souhaitaient voir HTTP survivre et devenir un vrai standard pluripartite, et nous n'étions absolument pas opposés à ce que Microsoft ou Netscape incorporent dans leurs produits notre moteur HTTP ou un quelconque autre composant de notre code si cela favorisait davantage l'érection de HTTP en tant que standard commun.

Ce degré d'ouverture comporte des risques. La licence ne contient aucune incitation destinée à encourager les entreprises à contribuer à améliorer le code, et donc le projet en retour. Il y a certainement eu dans l'histoire d'Apache des cas où des entreprises ont développé à partir d'Apache des approches que nous aurions voulues voir réincorporées. Mais une licence exigeant cela aurait peut-être a priori condamné de telles améliorations.

Tout cela signifie que, sur le plan stratégique, le projet a besoin d'un élan continu, et que les participants doivent réaliser davantage de bénéfices en contribuant au projet qu'en conservant par-devers eux leurs modifications. Cet état demeure délicat à maintenir, surtout si une entreprise décide d'augmenter le volume des efforts de développement qu'elle consacre à un projet dérivé et commence à douter du retour potentiel en proportion de sa contribution au projet. Les responsables s'exclament : « Nous menons à bien davantage de travail que tous les autres regroupés, et pourquoi devons-nous partager ? ». Je ne détiens pas de formule magique adaptée à ce scénario, mais affirme qu'en ce cas l'entreprise créatrice n'a probablement pas trouvé le meilleur moyen de susciter efficacement des contributions de tierces parties.

11.8.2 La licence publique de Mozilla (MPL)

L'équipe Mozilla de Netscape a mis au point la licence publique de Mozilla afin de protéger son projet. Cette dernière, parue plusieurs années après les autres, doit résoudre certains problèmes cruciaux posés par les licences BSD et GNU. Son style, dans le corps des groupes Open Source, ressemble surtout à celui de la licence BSD. Elles présentent toutefois deux différences majeures :

La MPL requiert que les modifications apportées à la « distribution » soient aussi publiées sous MPL, et restent ainsi intégrables au projet. « Distribution » désigne ici les fichiers du code source distribués. Cette importante disposition permet à une entreprise d'ajouter une interface à une bibliothèque de code propriétaire sans exiger que les autres bibliothèques de code soient aussi diffusées sous MPL, elle ne concerne que l'interface. Le logiciel ainsi couvert peut être plus ou moins inclus dans un environnement de produit commercial.

Plusieurs clauses de la MPL protègent à la fois le projet dans son ensemble et ses développeurs contre un brevet déposé sur le code contribué. Elle impose que l'entreprise ou la personne contribuant au projet abandonne tout droit découlant du code.

Il ne faut pas négliger l'importance de cette seconde clause, mais elle contient aussi aujourd'hui un gros défaut.

S'atteler à la question du brevet est une Très Bonne Chose. Une entreprise pourrait offrir le code à un projet afin d'essayer, après son intégration, de revendiquer ses droits pour son utilisation. Une telle stratégie commerciale pourrait paraître risible et serait très regrettable, mais malheureusement certaines sociétés n'ont pas encore saisi cela. Ainsi, la seconde clause interdit à quiconque de fournir subrepticement du code qu'il sait breveté, et de créer par là-même des ennuis à tous les autres participants.

Cela ne signifie pas qu'un tiers, détenteur d'un brevet, se manifeste ensuite. Aucun instrument légal n'offre de protection contre cela. En fait, je défendrais que cela doit faire l'objet d'une mission confiée à un nouveau service du bureau américain des brevets géré par le ministère du commerce et de l'industrie qui semble détenir le pouvoir d'attribuer certaines idées ou certains algorithmes et devraient donc être en mesure, réciproquement, de certifier qu'un code proposé est libre de droits, épargnant ainsi a priori au donateur tout tracas juridique.

Comme je l'ai dit ci-dessus, la licence MPL actuelle (décembre 1998) contient un défaut. Par essence, la section 2.2 précise (dans sa définition de « version du contributeur ») que le contributeur renonce aux droits sur toute partie de Mozilla et non seulement sur le code soumis. Cela ne semble pas fâcheux. De grandes entreprises collaboreraient ainsi sans contrainte.

Malheureusement, une certaine grosse entreprise détentrice de l'un des plus gros portefeuilles de brevets y voit un problème majeur. Non parce qu'elle a l'intention de demander un jour à Mozilla des royalties, ce qui paraîtrait fort téméraire. Mais elle est concernée car certaines parties de Mozilla utilisent des procédés dont elle détient les brevets grâce auxquels perçoit régulièrement d'importantes sommes d'argent. Elle ne souhaite donc pas renoncer à ses brevets sur le code de Mozilla, car les entreprises qui paient pour employer ces techniques incluraient alors le code de Mozilla correspondant dans leurs produits, sans leur servir de rente. Si la section 2.2 se référait simplement aux corrections contribuées plutôt qu'à l'ensemble du navigateur pour le renoncement aux brevets, ceci ne poserait pas de problème.

La licence MPL, si l'on écarte cette bizarrerie, demeure remarquablement solide. Exiger le retour des changements vers le « noyau » implique l'intégration des corrections essentielles de bogues et des améliorations, et laisse des entités commerciales développer des fonctionnalités à valeur ajoutée. La MPL reste peut-être la licence la plus adéquate pour des applications destinées à l'utilisateur final, où il est plus probable que les brevets posent problème, et où le dynamisme mène à la diversification des produits sous-jacents. Au contraire, la licence BSD est peut-être plus appropriée dans le cas d'un projet destiné à rester « invisible » ou de la réalisation de bibliothèques de fonctions essentielles, comme par exemple un système d'exploitation ou un serveur Web.

11.8.3 La licence publique GNU (GNU Public Licence, GPL)

Bien qu'à première vue elle n'incite guère au commerce, certains aspects de la licence GNU sont, croyez-le ou non, favorables à des objectifs commerciaux.

Fondamentalement, la GPL requiert la publication sous une même licence des améliorations, des dérivés, et même du code qui incorpore un code sous GPL. Des tenants du logiciel libre défendirent cette disposition « infectieuse » car elle constitue selon eux un moyen de s'assurer que le code libre le demeure et qu'aucune entité ne propose sa propre version du logiciel sans la rendre publique. Ceux qui placent leur logiciel sous GPL préfèrent priver le projet d'une contribution plutôt que de risquer de ne pouvoir l'utiliser librement. Cette approche présente un certain attrait d'ordre théorique, bien sûr, et certains affirment que le succès de Linux doit beaucoup à la GPL, car une récupération commerciale, très tentante, aurait sinon perturbé de façon décisive la cohérence de son développement.

Ainsi, à première vue, la GPL semble peu compatible avec tout objectif commercial lié au logiciel libre. Les modèles traditionnels grâce auxquels la plus-value du logiciel se matérialise sous forme d'argent paraissent a priori écartés. Elle pourrait toutefois offrir un moyen extraordinairement efficace d'établir une plate-forme interdisant à un concurrent de proposer la sienne, et donc de vous protéger lorsque vous vous déclarez premier fournisseur de biens et de services pour elle.

La relation entre Cygnus™ et GCC illustrent bien cela. L'entreprise Cygnus™ assure de fréquentes et nombreuses modifications de GCC en le portant vers différents types de matériels. Elle verse la plus grande partie de ces modifications, en accord avec la GPL, sous forme de contributions à la distribution de GCC, et les rend donc disponibles. Cygnus™ facture ses prestations de portage et de maintenance, et non le code lui-même. L'histoire de Cygnus™ et de sa prééminence sur ce marché en font une société de référence pour ce type de service.

Si un concurrent tentait de concurrencer Cygnus™, il lui faudrait redistribuer ses modifications sous GPL. Cela signifie que GCC ne lui offrirait aucune niche technique ou commerciale exploitable sans donner à Cygnus™ la même occasion d'en prendre avantage. Cygnus™ a créé une situation dans laquelle les concurrents ne peuvent la concurrencer sur le plan technique, à moins de dépenser beaucoup de temps et d'argent et d'utiliser une autre plate-forme que GCC.

La GPL peut aussi être utilisée à des fins commerciales en tant que sentinelle technologique, proposée avec une version non GPL du même logiciel. Examinons par exemple le cas d'un excellent programme de chiffrement des connexions TCP/ IP. Vous ne vous souciez pas des utilisations non commerciales ou même commerciales mais souhaitez percevoir des droits lorsque d'autres entreprises incluent le code dans un produit ou le redistribuent et réalisent des bénéfices. Si vous placez le code sous GPL, nul ne peut l'intégrer librement dans un produit commercial sans le placer sous GPL, et la plupart d'entre eux s'y refuseront. Cependant, si vous maintenez une branche séparée de votre projet non protégée par la GPL vous pouvez la breveter. Vous devez toutefois vous assurer que tout code apporté par des tiers est explicitement rendu disponible pour cette branche commerciale en déclarant que vous en restez seul auteur et restez libre d'y intégrer tout code soumis pour intégration dans la version sous GPL.

Cela offre à certaines entreprises un modèle commerciale viable. Transvirtual, à Berkeley, applique ce schéma à une machine virtuelle Java légère et à un projet de bibliothèque de classes. Selon certains le nombre de contributeurs attirés par un tel modèle s'avérera faible et les versions GPL et non GPL convergeront donc. À mon sens si vous traitez les contributeurs comme il se doit, peut-être même en leur offrant de l'argent ou tout autre forme de compensation (ils épaulent, après tout, votre approche commerciale), ce modèle devrait fonctionner.

Le champ d'action de la licence Open Source évoluera à mesure que les gens valideront les approches efficaces et écarteront les autres. Vous restez libre de mettre au point une nouvelle licence placée où bon vous semble dans le spectre disponible (représenté par BSD à ma droite et GPL à ma gauche). Ne négligez pas que ceux qui utilisent et étendent votre code se sentiront d'autant plus incités à continuer qu'ils se sentiront libres.

11.9 Les outils d'aide à la création de Projets à code ouvert

Le projet Apache dispose d'outils de conduite partagée du développement, bien entretenus et fort efficaces.

Le plus important parmi ceux-ci est CVS ou « Concurrent Versioning System » (un « Système de gestion des révisions »). Il s'agit d'une collection de programmes maintenant un répertoire de code partagé puis assurant sa mise à jour au gré des modifications en mémorisant les dates et les noms de leurs auteurs. De nombreuses personnes agissent ainsi simultanément sur le même programme sans se gêner mutuellement. Le processus de déverminage s'en trouve aussi facilité du fait que CVS rend possible d'examiner un à un tous les changements survenus afin de trouver l'origine exacte d'un bogue donné. Le code client de CVS, disponible pour toutes les plates-formes majeures, fonctionne tout-à-fait correctement via des lignes téléphoniques et sur des connexions longues distances. SSH en assure si nécessaire la confidentialité tout en gérant les habilitations.

Le projet Apache n'utilise pas seulement CVS pour mettre à jour le code source mais aussi pour mettre à jour le fichier STATUS dans lequel nous plaçons toutes les questions importantes et les descriptions commentées et raisonnées d'innovations majeures, et même les résultats de scrutins liés à chaque question-clé. CVS enregistre aussi les bulletins de vote émis lors des prises de décisions

du groupe, maintient les documents de notre site Web, gère la documentation de développement, etc. En bref, il constitue à la fois notre coffre-fort et notre livret de gestion du savoir. Sa simplicité peut effrayer (la plupart des logiciels de cette catégorie sont chers et très touffus) mais, en pratique, est l'une de ses plus grandes qualités. Tous ses composants, côté serveur comme côté client, sont libres.

Un projet aux sources libres doit offrir un solide ensemble de forums de discussion à ses développeurs et utilisateurs. Le logiciel utilisé importe peu (nous employons Majordomo, mais Ezmlm ou Smartlist ou n'importe quel équivalent rendrait autant de services). Il faut consacrer une liste à chaque sous-groupe cohérent de développeurs, de sorte qu'ils sélectionnent eux-même les thèmes traités et restent proches. Créer, de plus, une liste séparée pour chaque projet grâce à laquelle le serveur CVS diffuse des avis de changements effectués est aussi très important car cela offre aux participants le moyen d'examiner au fur et à mesure les modifications effectuées. Cette approche invite tout un chacun à respecter les standards de codage et facilite la recherche des bogues. Mieux vaut séparer les listes des utilisateurs de celles des développeurs, et peut-être même, si le nombre de participants augmente, le noyau des développeurs activistes des autres. Préserver les archives des listes disponibles publiquement afin de laisser les utilisateurs y chercher des réponses à leurs questions.

Un projet bien mené comprend aussi un outil de suivi des bogues et des versions. Dans le cadre du projet Apache nous utilisons un outil GNU appelé GNATS. Il gère parfaitement plus de 3000 rapports de bogues, et les corrections correspondantes. Vous voulez trouver un outil grâce auquel de nombreuses personnes pourront répondre à des rapports de bogues, certaines en se spécialisant sur des éléments donnés du projet, capable de diffuser des informations par la messagerie électronique et d'interagir grâce à cette dernière plutôt qu'exclusivement via le Web. Adoptez un outil facile à exploiter et automatisable, afin que les développeurs répondent de façon aisée (la plupart d'entre eux n'apprécient guère cette phase) et aussi de sorte qu'ils retrouvent facilement tout bogue déjà découvert. Cette base de données deviendra de fait le dépôt de la connaissance périphérique à votre projet. Une telle base de données doit permettre de répondre à la traditionnelle question « Pourquoi tel comportement est-il une fonctionnalité et non un bogue ? »

L'approche Open Source n'est pas la panacée éliminant à l'avance toute menace. Non seulement parce que des conditions favorables restent toujours nécessaires, mais aussi parce que la gestion d'un développement en cours exige une extraordinaire somme de travail. Dans de nombreux cas vous devrez, en tant que promoteur d'un nouveau projet, agir un peu comme le docteur Frankenstein, mélangeant ici des éléments et appliquant là une tension électrique afin de donner vie à votre création. Bonne chance.

Chapitre 12

La définition de l'Open Source

12.1 Introduction

L'utilisateur-type d'ordinateurs possède des quantités de logiciels acquis dans le passé mais qu'il n'utilise plus. Il a peut-être acheté un ordinateur plus performant ou changé de constructeur, ce qui lui interdit d'employer ses programmes. Ils sont peut-être devenus obsolètes, ou ne répondent plus à ses besoins. Il a peut-être acheté plusieurs ordinateurs et ne souhaite pas débourser quoi que ce soit pour obtenir un deuxième exemplaire du même logiciel. Quelle qu'en soit la raison, le logiciel lui a coûté de l'argent voici quelques années et ne remplit plus son rôle. Cette situation est-elle inévitable ?

Que penseriez-vous d'une situation dans laquelle vous auriez le droit d'obtenir une mise à jour dès que cela s'avère nécessaire ? Que penser d'une situation dans laquelle, en passant l'architecture Mac à l'architecture compatible IBMPC, on pourrait changer gratuitement les versions des logiciels ? Et si, quand par malheur le logiciel ne fonctionne pas ou ne remplit pas tous vos besoins, vous pouviez le faire améliorer ou le réparer vous-même ? Peut-on imaginer qu'un logiciel soit encore maintenu et disponible après la faillite de son éditeur ? Que l'on puisse les utiliser sur la station de travail installée au bureau, sur l'ordinateur personnel du domicile, et sur son portable, plutôt que d'être limité à une seule machine ? Si tel était le cas, vous continueriez sans doute à utiliser les logiciels achetés voici des années. Ce sont là quelques-uns des droits que l'open source vous garantit.

L'open source est une déclaration des droits de l'utilisateur d'ordinateur. Elle définit certains droits que la licence d'un logiciel doit lui garantir afin d'être qualifiée d'open source. Ceux qui ne proposent pas des programmes open source estiment que les autres leur opposent une concurrence de plus en plus rude, car ils n'ont pas compris que les utilisateurs prennent enfin conscience de leurs droits, et donc les revendiquent. Des programmes comme ceux qui constituent le système d'exploitation GNU/Linux et le navigateur web de Netscape ont acquis une grande popularité en prenant la place de logiciels protégés par des licences plus restrictives. Les sociétés qui utilisent du logiciel open source profitent des avantages liés à un modèle de développement très réactif, souvent mis en place par plusieurs sociétés qui coopèrent, et dont une grande partie du travail est fournie par des indépendants qui ont tout simplement besoin d'une amélioration qui réponde mieux à leurs besoins.

Les volontaires qui ont créé des produits comme Linux n'existent, et les sociétés ne peuvent coopérer, que grâce aux droits fournis par l'open source. Le programmeur moyen se sentirait lésé s'il investissait énormément de travail dans un programme, pour constater que le propriétaire de ce dernier se contente de vendre la version améliorée sans rien lui proposer en retour. Mais ces mêmes programmeurs n'ont pas peur d'apporter des contributions à l'open source, car ils savent que les droits suivants leur sont alors garantis :

- le droit de faire des copies du programme, et de les distribuer,
- le droit d'accéder au code source du logiciel, un préliminaire nécessaire pour pouvoir y apporter des modifications,
- le droit d'améliorer le programme.

Ces droits comptent pour celui qui contribue à améliorer des logiciels car ils maintiennent tous les développeurs au même niveau. Quiconque le souhaite a le droit de vendre un programme open source, aussi les prix seront-ils bas

et le développement destiné à conquérir de nouveaux marchés sera-t-il rapide. Quiconque investit de son temps à construire des connaissances sous la forme d'un programme open source peut fournir de l'aide sur ce dernier, et cela fournit aux utilisateurs la possibilité de mettre en place leur propre assistance technique, ou de choisir parmi des assistances techniques en concurrence les unes avec les autres. Tout programmeur peut spécialiser un programme open source pour le rendre utilisable dans des marchés spécifiques afin d'atteindre de nouveaux consommateurs. Ceux qui se lancent dans de telles entreprises ne sont pas tenus d'acquitter des royalties ni la moindre licence.

La raison de la réussite de cette stratégie qui semble s'inspirer du communisme, alors que le communisme est lui-même un échec patent dans le monde entier, vient du fait que l'économie de l'information est fondamentalement différente de l'économie qui règle la consommation des autres produits. On peut copier une information telle qu'un programme d'ordinateur pour un prix dérisoire. L'électricité qu'il en coûte ne dépasse par l'eurocent, pas plus que l'utilisation de l'équipement nécessaire à cet acte. En comparaison, on ne peut pas recopier une miche de pain sans dépenser une livre de farine.

12.2 L'histoire

Le concept de logiciel libre est ancien. Quand les ordinateurs sont entrés dans les universités, c'était en tant qu'outils de recherche. Les logiciels étaient librement passés de laboratoire en laboratoire, et les programmeurs étaient payés pour le fait de programmer, et non pour les programmes en eux-mêmes. Ce n'est que plus tard, quand les ordinateurs sont entrés dans le monde des affaires, que les programmeurs ont commencé à subvenir à leurs besoins en limitant les droits associés à leurs logiciels et en en taxant chaque copie. Richard Stallman répand les concepts politiques liés au Logiciel Libre depuis 1984, année où il créa la Free Software Foundation et son projet GNU. La prémisse de M.Stallman est que tout le monde devrait jouir de plus de liberté, et savoir l'apprécier. Il a dressé la liste des droits dont tout utilisateur doit selon lui jouir, et les a codifiés au sein de la licence publique générale de GNU, ou GPL. M.Stallman a intitulé, non sans malice, cette licence « gauche d'auteur » (copyleft), car au lieu d'interdire, elle donne le droit de copier. M.Stallman a lui-même développé de féconds travaux en matière de logiciels libres, tels que le compilateur de langage C du projet GNU, et GNU Emacs, un éditeur disposant d'attraits tels que certains en parlent comme d'une religion. Ses travaux ont inspiré de nombreux autres développeurs à proposer du logiciel libre selon les conditions de la GPL. Même si elle n'est pas promue avec la même ferveur libertaire, la définition de l'open source reprend de nombreuses idées de M.Stallman, et on peut la considérer comme un texte dérivé de ses propres travaux.

La définition de l'open source a vu le jour en tant que document présentant les grandes lignes du logiciel libre au sens de la distribution Debian de Gnu/ Linux. Debian, un système Linux de la première heure, toujours populaire de nos jours, était entièrement constitué de logiciels libres. Cependant, comme la « gauche d'auteur » n'était pas la seule licence qui prétendît couvrir du logiciel « libre », Debian avait des difficultés à tracer la limite entre ce qui est libre et ce qui ne l'est pas, et n'avait jamais clairement indiqué les règles qu'elle appliquait pour déterminer si un logiciel était libre ou non. Je dirigeais alors le projet

Debian, et j'ai résolu ce problème en proposant le Contrat social de Debian et les Lignes de conduite en matière de logiciel libre de Debian en juillet 1997. De nombreux développeurs Debian ont proposé des critiques et des améliorations, que j'incorporais peu à peu dans les documents. Le Contrat social documentait l'intention de la distribution Debian de ne composer leur système que de logiciels libres, et les Lignes de conduite en matière de logiciel libre permettaient de classer facilement le logiciel en deux catégories, « libre » ou non, en comparant la licence du logiciel aux lignes de conduite.

La ligne de conduite de Debian reçut les louanges de la communauté du logiciel libre, et en particulier des développeurs de Linux, qui menaient alors leur propre révolution du logiciel libre en développant le premier système d'exploitation libre utilisable. Quand la société Netscape a décidé de faire de son navigateur pour le web un logiciel libre, elle a contacté EricRaymond, véritable Margaret Meade [1] du logiciel libre qui a écrit plusieurs articles anthropologiques expliquant ce phénomène et détaillant la culture qui a peu à peu germé autour de lui, premiers travaux sur le sujet qui ont fait la lumière sur ce phénomène auparavant peu connu. La direction de la société Netscape fut impressionnée par l'essai de M.Raymond intitulé « La cathédrale et le bazar », chronique de la réussite du développement de logiciels libres, qui n'utilise que des contributeurs volontaires non rémunérés, et l'a engagé en tant que conseiller, sous un accord de non divulgation, pendant qu'elle développait une licence pour son propre logiciel libre. M.Raymond leur a clairement indiqué qu'il fallait que cette licence suive les lignes de conduite de Debian pour être prise au sérieux en tant que licence de logiciel libre.

J'ai rencontré M.Raymond de façon fortuite à la conférence des Hackers, réunion peu conventionnelle de programmeurs créatifs, réservée aux invités. Nous avions traité de divers sujets par courrier électronique. Il m'a contacté en février 1997 en me présentant l'idée de l'open source. Il regrettait alors que les hommes d'affaires, conservateurs, soient effrayés par le mouvement de liberté initié par M.Stallman, qui rencontrait par contraste un écho très favorable auprès des programmeurs les plus libertaires. Il pensait que cela étouffait le développement de Linux dans le monde des affaires tout en encourageant sa croissance dans le milieu de la recherche. Il a rencontré les hommes d'affaires de l'industrie balbutiante qui se formait autour de Linux, et ils ont conçu ensemble un programme pour faire la campagne du concept de logiciel libre auprès de ceux qui portent des cravates. LarryAugustin, de VA Research, et SamOckman (qui a quitté VA un peu plus tard pour créer la société Penguin Computing) ont participé à l'affaire, ainsi que d'autres personnes, que je n'ai pas la chance de connaître.

Quelques mois avant de lancer le projet de l'open source, j'avais conçu l'idée de l'OpenHardware [2], concept similaire, mais traitant des périphériques matériels et de leurs interfaces plutôt que de programmes et de logiciels. L'OpenHardware n'a pas rencontré à ce jour le même succès que l'open source, mais il suit son bonhomme de chemin et vous trouverez toutes informations le concernant à l'adresse http://www.openhardware.org/.

M.Raymond estimait que les Lignes de conduite du logiciel libre selon Debian était le document indiqué pour définir l'open source, mais qu'il leur fallait

1. célèbre anthropologue américaine
2. matériel dont les spécifications sont ouvertes

un nom plus général et en extirper toute référence spécifique à la distribution Debian. J'ai édité les Lignes de conduite pour mettre en place la définition de l'open source. J'avais formé une corporation pour Debian, intitulée SPI (logiciel d'intérêt public), et j'ai proposé d'enregistrer une marque déposée pour l'open source de telle sorte qu'on puisse réserver son utilisation aux logiciels qui répondent à la définition. M.Raymond étant d'accord avec cette idée, j'enregistrai la marque de certification, ce qui est une forme particulière de marque déposée, à appliquer aux produits des autres. Un mois après que j'aie enregistré cette marque, il était devenu clair que SPI n'était pas le lieu d'hébergement idéal pour la marque open source, et j'en ai transféré la propriété à M.Raymond. Depuis, nous avons créé l'open sourceInitiative, organisation dont le seul but est de gérer la campagne de l'open source et sa marque de certification. Au moment de cet écrit, son conseil d'administration compte six membres choisis parmi des contributeurs renommés du monde du logiciel libre, et cherche à atteindre la taille d'environ dix membres.

Lors de son lancement, la campagne de l'open source a suscité de nombreuses critiques, même au sein des troupes dévouées à Linux, qui avaient déjà accepté le concept du logiciel libre. Nombreux sont ceux qui signalèrent que le terme « open source » était déjà employé dans l'industrie de XXX . D'autres pensaient que le mot « Open » était déjà utilisé à toutes les sauces. D'autres, simplement, préféraient le terme Free Software, établi. J'ai répondu que le galvaudage du terme « Open » ne pourrait jamais causer autant de tort que l'ambiguïté, dans la langue anglaise, du terme « Free » la liberté ou la gratuité[3], la notion de « gratuité » étant celle que le monde commercial, en informatique, met le plus souvent en avant. Richard Stallman a plus tard pris ombrage du fait que la campagne ne mettait pas suffisamment l'accent sur la liberté, et que le mouvement de l'open source gagnant en popularité, son rôle dans la genèse du logiciel libre, ainsi que celui de la Free Software Foundation qu'il a créée, étaient passés sous silenceil s'est plaint d'être « effacé des livres d'histoire ». La situation s'est aggravée par la tendance qu'ont les industriels de comparer MM.Raymond et Stallman comme s'ils défendaient des philosophies contradictoires, sans comprendre qu'en réalité ils ne font qu'utiliser des méthodes différentes pour promouvoir le même concept. J'ai probablement contribué à jeter de l'huile sur le feu en opposant MM.Stallman et Raymond dans des débats tenus à l'occasion des manifestations Linux Expo et open source Expo. Leurs confrontations étaient si appréciées que le journal en ligne Salon est allé jusqu'à publier un débat qu'ils ont tenu par courrier électronique, sans jamais avoir eu l'intention de le publier. Quand nous en fûmes rendus là, j'ai demandé à M.Raymond de calmer un jeu auquel il n'avait jamais eu l'intention de participer.

Lors de l'écriture de la définition de l'open source, nombreux déjà étaient les programmes qui y satisfaisaient. Mais un problème se posait dans le cas des programmes qui n'y correspondaient pas, tout en plaisant aux utilisateurs.

3. en anglais, le mot « free » signifie à la fois « libre » et « gratuit », alors que le mot « open » signifie « ouvert ». Consulter les essais de Richard Stallman et d'Eric Raymond pour obtenir plus d'informations à ce sujet.

12.3 KDE, Qt, et Troll Tech

Il faut ici traiter du groupe de développement de KDE et de la société Troll Tech car ils ont tenté de placer un produit non conforme à l'open source dans l'infrastructure de Linux, et se sont heurtés à une résistance qu'ils n'avaient pas su prévoir. Le tollé général et la menace d'un remplacement de leur produit par une solution open source ont finalement convaincu la société Troll Tech d'adopter une nouvelle licence pleinement conforme à l'open source. C'est un exemple intéressant de l'acceptation enthousiaste par la communauté de la définition de l'open source, si forte qu'il fallait que la société Troll Tech s'y plie si elle souhaitait que son produit soit bien accueilli.

KDE représente la première tentative d'un bureau graphique libre pour Linux. Les applications de KDE étaient par elles-mêmes protégées par la GPL, mais elles dépendaient d'une bibliothèque graphique propriétaire, appelée Qt, et développée par la société Troll Tech. Les conditions de la licence de Qt en interdisaient la modification ou l'utilisation en relation avec tout logiciel d'affichage graphique autre que le vieillissant X Window System. Pour ces autres utilisations, il fallait acheter une licence de développeur qui s'élevait à 1500USD. La société Troll Tech fournissait des versions de Qt pour les systèmes MS-Windows et Macopen source, et c'était là sa première source de revenus. La prétendue « licence libre » pour les systèmes fonctionnant sous « X » avait pour objectif de susciter des contributions de la part des développeurs de Linux en matière de démonstrations, exemples, et autres accessoires, qu'ils revendraient ensuite dans le cadre de leurs produits pour MS-Windows et Macopen source, qui n'étaient pas donnés.

Malgré la clarté des problèmes posés par la licence de Qt, la perspective de disposer d'un bureau graphique sous Linux était si séduisante que de nombreux utilisateurs étaient prêts à fermer les yeux sur le fait que ce n'était pas vraiment une licence de type open source. Les défenseurs de l'open source remettaient KDE en question car ils estimaient que les développeurs de KDE tentaient d'estomper la définition du logiciel libre pour y inclure des éléments qui ne le sont que partiellement, comme Qt. Ce à quoi ces derniers rétorquaient que leurs programmes étaient open source, même s'il n'en existait aucune version exécutable qui ne nécessitât pas de bibliothèque non open source. D'autres et moi pensions que les applications de KDE n'étaient que des fragments open source de programmes non open source, et qu'on ne pourrait pas qualifier KDE d'open source avant de disposer d'une version open source de Qt.

Les développeurs de KDE ont tenté de résoudre en partie le problème posé par la licence de KDE en négociant un accord sur une base Qt libre pour KDE avec Troll Tech, selon lequel ces derniers et le groupe KDE contrôleraient ensemble les publications de la version libre de Qt, et Troll Tech publierait Qt selon les conditions d'une licence conforme à l'open source si la société devait être rachetée ou faire banqueroute.

Un autre groupe de programmeurs démarra alors le projet GNOME, un concurrent de KDE pleinement open source, qui visait à être plus sophistiqué et à proposer plus de fonctionnalités, pendant qu'un troisième groupe démarrait un projet intitulé Harmony dans le but de produire un clone de Qt pleinement open source, qui permettrait de faire fonctionner KDE. Alors que le projet GNOME était applaudi de tous et que le projet Harmony était sur le point d'aboutir,

la société Troll Tech a compris que Qt ne rendrait aucun service au marché de Linux tant que sa licence demeurerait en l'état. Troll Tech a publié une licence pleinement open source pour Qt, désamorçant ainsi le conflit, et laissant retomber le projet Harmony comme un soufflé. Mais le projet GNOME suit son cours, et cherche maintenant à surclasser KDE en termes de fonctionnalités et de sophistication, plutôt qu'en matière de licence.

Avant de publier sa licence open source, la société Troll Tech m'en a fourni une copie pour que je l'examine, en me demandant de n'en rien laisser filtrer jusqu'au moment où elle l'annoncerait. Trop enthousiaste d'enterrer la hache de guerre avec le groupe KDE et victime d'un aveuglement passager que je me reprocherais par la suite, j'ai fait une pré-annonce de leur future licence avec huit heures d'avance sur une liste de diffusion consacrée à KDE. Ce courrier fut presque immédiatement repris, à mon grand regret, par les principaux sites d'information en ligne, dont Slashdot.

La nouvelle licence de Troll Tech a ceci de remarquable qu'elle tire avantage d'un point faible de la définition de l'open source qui accorde aux fichiers de modifications (patches) un statut particulier. Je souhaiterais le corriger dans une prochaine version de la définition de l'open source, mais ce nouveau texte excluerait alors de nouveau la bibliothèque Qt.

Au moment où j'écris ces lignes, le nombre de défenseurs de l'open source augmente exponentiellement. Les récents apports des sociétés IBM et Ericsson au mouvement de l'open source ont fait la une. On compte deux distributions de Linux, Yggdrasil et Debian, qui diffusent des systèmes Linux conformes en tous points à l'open source, sans pour autant se priver d'une grande richesse de logiciels, tandis que plusieurs autres distributeurs, comme Red Hat, n'en sont pas loin. Quand le projet GNOME sera disponible[4], on disposera enfin d'un système d'exploitation avec bureau à interface utilisateur graphique, digne de concurrencer MS-WindowsNT et entièrement open source.

12.4 Analyse de la définition de l'open source

Dans cette section, je présenterai le texte complet de la définition de l'open source, en y entrelaçant des commentaires (en hors-texte). Vous en trouverez la version canonique à l'adresse http://www.opensource.org[5].

Des coupeurs de cheveux en quatre m'ont fait remarquer que la définition de l'open source renfermait quelques ambiguïtés. J'ai reculé l'instant d'une révision car ce texte date d'à peine un an et j'aimerais qu'il soit considéré comme un texte stable. Dans l'avenir, j'y incorporerai des modifications mineures sur la forme, qui n'auront que très peu d'incidence sur le fond.

12.4.1 Version 1.0 de la définition d'open source

« open source » implique plus que la simple diffusion du code source. La licence d'un programme « open source » doit correspondre aux critères suivants :

> On remarque que la définition de l'open source n'est pas une licence de logiciel en soi. C'est une spécification de ce qu'on autorise aux

4. la première version stable en a été annoncée en février 1999
5. une version en français est disponible

licences de logiciels pour qu'elles méritent le nom d'open source. La définition de l'open source n'a pas pour vocation d'être un document juridique. Le fait qu'on la retrouve dans des licences de logiciels, comme celle du Linux Documentation Project (projet de documentation pour Linux), m'a fait envisager d'en écrire une version plus rigoureuse, qui serait appropriée pour une telle utilisation.

Pour qu'un logiciel puisse être qualifié d'open source, il faut que toutes les conditions suivantes soient remplies, en même temps, et dans tous les cas. Il faut par exemple qu'elles s'appliquent aux versions dérivées d'un programme aussi bien qu'au programme original. Il ne suffit pas de n'en appliquer que quelques unes et pas d'autres, et il ne suffit pas de ne les appliquer que dans certaines périodes. Comme j'ai dû ferrailler pour contrer des interprétations particulièrement naïves de la définition de l'open source, je serai tenté d'ajouter : cela vous concerne !

12.4.2 Libre redistribution.

La licence ne doit pas interdire de vendre ou de donner le logiciel en tant que composant d'une distribution d'un ensemble contenant des programmes de diverses origines. La licence ne doit pas soumettre cette vente à l'acquittement de droits, quels qu'ils soient.

> Cela signifie qu'on peut faire autant de copies du logiciel qu'on le souhaite, et les vendre ou les donner, sans devoir donner d'argent à qui que ce soit pour bénéficier de ce privilège.
>
> La formule « composant d'une distribution d'un ensemble contenant des programmes de diverses origines » avait pour but de combler une lacune de la licence Artistic (artistique), dont personnellement je trouve qu'elle manque de rigueur, qui a été mise au point pour Perl, à l'origine. De nos jours, presque tous les programmes qui en font usage sont aussi proposés selon des conditions de la GPL. C'est pourquoi cette clause n'est plus nécessaire et disparaîtra peut-être d'une prochaine version de la définition de l'open source.

12.4.3 Code source

Le programme doit inclure le code source, et la distribution sous forme de code source comme sous forme compilée doit être autorisée. Quand un produit n'est pas distribué avec le code source correspondant, il doit exister un moyen clairement indiqué de télécharger ce code source, depuis l'Internet, sans frais supplémentaires. Le code source est la forme la plus adéquate pour qu'un programmeur modifie le programme. Il n'est pas autorisé de proposer un code source rendu difficile à comprendre. Il n'est pas autorisé de proposer des formes intermédiaires, comme ce qu'engendre un préprocesseur ou un traducteur automatique.

Le code source est un préliminaire nécessaire à la correction ou la modification d'un programme. L'intention est ici de faire en sorte que le code source soit distribué aux côtés de la version initiale et de tous les travaux qui en dériveront.

12.4.4 Travaux dérivés

La licence doit autoriser les modifications et les travaux dérivés, et leur distribution sous les mêmes conditions que celles qu'autorise la licence du programme original.

Le logiciel est de peu d'utilité à qui ne peut assurer son évolution (correction des bogues, portage vers de nouveaux systèmes, amélioration), et il est nécessaire pour cela de le modifier. L'intention est ici d'autoriser tous types de modifications. Il faut autoriser qu'un travail dérivé soit distribué sous les mêmes conditions de licence que le travail original. Cependant, on n'exige pas que le producteur d'un travail dérivé utilise les mêmes conditions de licence mais on impose de leur laisser la possibilité de le faire. Les différentes licences traitent ce problème de manières diverses : la licence BSD vous autorise à privatiser vos modifications, alors que la GPL vous l'interdit.

Certains auteurs de logiciels craignent que cette clause n'autorise des gens peu scrupuleux à modifier leur logiciel de sorte à mettre dans l'embarras l'auteur original du logiciel. Ils ont peur qu'un individu mal intentionné ne fasse réagir le logiciel de manière incorrecte en laissant croire que l'auteur original était un programmeur de piètre qualité. D'autres craignent que le logiciel ne soit modifié pour des utilisations criminelles, par l'addition de fonction jouant le rôle de cheval de Troie ou de techniques interdites dans certains pays ou régions, comme la cryptographie. Mais de telles actions tombent sous le coup des lois. On pense souvent à tort que les licences de logiciels devraient tout spécifier, y compris des détails comme « n'utilisez pas ce logiciel pour commettre un crime. ». Mais aucune licence n'a d'existence en dehors d'un corpus de lois civiles et pénales. Considérer qu'une licence peut s'affranchir des lois en vigueur est aussi idiot que considérer qu'un document rédigé en français puisse s'affranchir du dictionnaire, auquel cas aucun des mots utilisés n'aurait la moindre signification arrêtée.

12.4.5 Intégrité du code source de l'auteur

La licence ne peut restreindre la redistribution du code source sous forme modifiée que si elle autorise la distribution de fichiers patch aux côtés du code source dans le but de modifier le programme lors de sa construction.

> Certains auteurs craignaient que d'autres ne distribuent le code source enrichi de modifications qui pourraient être perçues comme relevant du travail de l'auteur original, en donnant une mauvaise image de lui. Cette clause leur donne la possibilité d'imposer que les modifications soient bien distinctes de leur propre travail, sans pour autant interdire toute modification. Certaines personnes trouvent inesthétique le fait que les modifications encourent le risque de devoir être distribuées sous la forme d'un fichier de modifications (patch) distinct du code source, alors même que des distributions de Linux comme Debian ou Red Hat font usage d'une telle procédure pour mettre en place les modifications qu'elles apportent aux programmes qu'elles distribuent. Il existe des programmes qui fondent directement les modifications au sein du code source principal, et on peut faire en sorte qu'ils soient exécutés automatiquement lors de l'extraction d'un paquetage de code source. C'est pourquoi une telle clause ne devrait pas créer de grandes privations.
>
> Vous remarquerez aussi que cette clause stipule que dans le cas des fichiers de modifications, la modification n'a lieu qu'au moment de la construction. La licence publique de Qt exploite cette lacune pour imposer une licence différente, quoique plus permissive, en matière de fichiers de modifications, en contradiction avec la section3 de la définition de l'open source. Il existe le projet de corriger cette lacune dans la définition sans pour autant faire perdre à la licence Qt son état de licence open source.

La licence doit explicitement permettre la distribution de logiciel construit à partir du code source modifié. Elle peut exiger que les travaux dérivés portent un nom différent ou un numéro de version distinct de ceux du logiciel original.

> Cela signifie que la société Netscape, par exemple, peut insister sur le fait qu'elle seule a le droit de donner à une version du programme le nom de Netscape Navigator (tm), alors que les versions libres du programme doivent porter un nom comme Mozilla ou autre chose encore.

12.4.6 Pas de discrimination sur l'identité de l'utilisateur

La licence ne doit opérer aucune discrimination à l'encontre de personnes ou de groupes de personnes.

> Une licence proposée par les Régents de l'université de Californie, à Berkeley, interdisait qu'un programme de conception de circuits électroniques soit employé par les forces de police de l'Afrique du Sud. Ce sentiment avait beau être généreux du temps de l'apartheid, cette clause n'a plus grand sens de nos jours. Certaines personnes sont toujours

coincées avec le logiciel qu'elles ont acquis sous cette condition, dont les versions dérivées doivent elles aussi porter la même restriction. Les licences open source ne doivent rien renfermer de tel, quelle que soit la générosité qui dicte de telles intentions.

12.4.7 Pas de discrimination sur le domaine d'application

La licence ne doit pas limiter le champ d'application du programme. Par exemple, elle ne doit pas interdire son utilisation dans le monde des affaires ou dans le cadre de la recherche génétique.

Votre logiciel doit pouvoir être utilisé aussi bien par une clinique qui pratique des avortements que par une organisation militant contre le droit à l'avortement. Ces querelles politiques relèvent de l'Assemblée Nationale, et non pas des licences de logiciels. Cette exigence choque beaucoup certaines personnes !

12.4.8 Distribution de la licence

Les droits attachés au programme doivent s'appliquer à tous ceux à qui le programme est transmis sans que ces parties doivent remplir les conditions d'une licence supplémentaire.

La licence doit s'appliquer automatiquement, sans exiger une quelconque signature. Malheureusement, on ne dispose d'aucun précédent juridique solide en matière de validité d'une licence applicable sans signature, quand elle passe d'une seconde à une tierce personne. Cependant, cet argument considère que la licence fait partie du droit du contrat, alors que certains argumentent qu'elle relève du droit du copyright, où on trouve des cas de jurisprudence en matière de licences ne requérant pas de signature. Il y a fort à parier que ce débat aura lieu en cour de justice d'ici quelques années, si l'on en juge par l'emploi sans cesse croissant de ce type de licences et par l'essor fulgurant du mouvement de l'open source.

12.4.9 Pas de spécificité

Les droits attachés au programme ne doivent pas dépendre de l'intégration du programme à un ensemble logiciel spécifique. Si le programme est extrait de cette distribution et utilisé ou distribué selon les conditions de la licence du programme, toutes les parties auxquelles le programme est redistribué doivent bénéficier des droits accordés lorsque le programme est au sein de la distribution originale de logiciels.

Cela par exemple signifie que vous ne pouvez pas contraindre un produit identifié en tant qu'open source à être utilisé en tant que composant d'une distribution particulière

de Linux. Il doit rester libre, même séparé de l'ensemble de logiciels avec lequelle il a été fourni.

12.4.10 Pas de contamination d'autres logiciels

La licence ne doit pas restreindre d'autres logiciels distribués avec le programme qu'elle protège. Par exemple, elle ne doit pas exiger que tous les programmes distribués grâce au même medium soient des logiciels « open source ».

Une version de GhostScript (programme de rendu de PostScript) exige que le support sur lequel est distribué ce programme ne contienne que des logiciels libres. Les licences open source ne permettent pas cela. Heureusement, l'auteur du programme GhostScript distribue une autre version (un peu plus ancienne) de ce programme, sous une licence vraiment open source.

Remarquez la différence entre dérivation et agglomération. La dérivation est le fait qu'un programme renferme en son sein une portion d'un autre programme. L'agglomération est le fait de proposer deux programmes sur le même CD-ROM. Cette section de la définition de l'open source traite de l'agglomération, pas de la dérivation. La section4 traite de cette dernière.

12.4.11 Exemples de licences

Les licences suivantes sont des exemples de licences que nous jugons conformes à la définition de l'« open source » : GNUGPL, BSD, XConsortium, et Artistic. C'est aussi le cas de la MPL.

Nous aurions beaucoup de problèmes si l'une de ces licences devait être modifiée de sorte qu'elle ne relève plus de l'open sourceil nous faudrait publier immédiatement une version révisée de la définition de l'open source. Ce paragraphe relève plus d'un texte d'explication que de la définition par elle-même.

12.5 Analyse des licences et de leur concordance avec la définition de l'open source

Pour comprendre la définition de l'open source, il faut examiner de quelle manière certaines licences communes en remplissent ou non les critères.

12.5.1 Domaine public

Une idée reçue incorrecte veut que la plupart des logiciels libres relèvent du domaine public[6]. La raison en est tout simplement que l'idée du logiciel libre

6. Le strict équivalent de la notion de domaine public, dans l'acception américaine de public domain, n'existe pas en France où la propriété intellectuelle est inaliénable.

ou de l'open source est difficile à appréhender pour de nombreuses personnes qui considèrent donc que ces programmes sont du domaine public car c'est là le concept le plus proche qu'ils arrivent à comprendre. Ces programmes disposent clairement, cependant, d'un copyright, et sont protégés par une licence. Il se trouve simplement que cette licence accorde plus de droits aux utilisateurs qu'ils n'en ont l'habitude.

Un programme du domaine public est un programme sur lequel son auteur a délibérément choisi de ne pas faire valoir ses droits. On ne peut pas vraiment dire qu'il est assorti d'une licence ; quiconque peut l'utiliser comme bon lui semble, car quiconque peut le traiter comme s'il lui appartenait. On peut même assujettir un programme du domaine public à une nouvelle licence, en ôtant cette version modifiée du domaine public, ou en ôter le nom de l'auteur et traiter le programme comme son propre travail.

Si vous travaillez beaucoup sur un programme du domaine public, envisagez le fait d'y apposer votre propre copyright et de lui appliquer une nouvelle licence. Si vous ne souhaitez pas, par exemple, qu'un tiers le modifie d'une manière qu'il gardera secrète ensuite, appliquez à votre version du programme la GPL, ou une licence similaire. La version de laquelle vous êtes parti se trouvera encore dans le domaine public, mais votre propre version sera placée sous une licence que les autres devront prendre en compte s'ils veulent l'utiliser ou en proposer des travaux dérivés.

Vous pouvez facilement capter (rendre privé) un programme du domaine public en déclarant un copyright et en lui appliquant votre propre licence, ou simplement en déclarant « tous droits réservés. »

12.6 Les licences de logiciels libres en général

Si vous disposez d'un collection de logiciels libres telle qu'un disque renfermant une distribution de GNU/Linux, vous pensez peut-être posséder les programmes qui se trouvent sur ce disque. Ce n'est pas exactement vrai. Les programmes sous copyright appartiennent au détenteur du copyright, même lorsqu'ils sont protégés par une licence open source telle que la GPL. La licence du programme vous octroie certains droits, et d'autres droits vous sont acquis selon la définition d'« utilisation raisonnable » dans le droit du copyright. Il est important de remarquer qu'un auteur n'est pas limité à proposer un programme sous une seule licence. On peut protéger un programme par la GPL et en vendre en même temps une version sous une licence commerciale et non open source. C'est exactement cette tactique qu'emploient de nombreuses personnes, qui veulent qu'un programme soit open source tout en souhaitant gagner de l'argent à partir de ce dernier. Ceux qui ne souhaitent pas d'une licence open source encourent le risque de devoir rémunérer un tel privilège, assurant par là même un revenu à l'auteur. Toutes les licences que nous allons maintenant examiner ont un point en commun : aucune ne fournit la moindre garantie. L'intention est de protéger le propriétaire du logiciel de toute poursuite en justice relative à son programme. Les programmes étant souvent offerts sans contrepartie financière, c'est là une exigence raisonnablele programme ne garantit pas un revenu suffisant pour que son auteur puisse acquitter une assurance et les frais engagés en cas de poursuite en justice. Si les auteurs de logiciels libres perdent ce droit de ne fournir aucune garantie et sont poursuivis en justice suite aux effets des

programmes qu'ils ont écrits, ils cesseront d'écrire des logiciels libres. C'est dans notre avantage d'utilisateurs que d'aider les auteurs à protéger ce droit.

12.6.1 La licence publique générale de GNU

La GPL est autant un manifeste politique qu'une licence de logiciel, et une grande partie de ce texte est dévolue à justifier les motifs qui ont poussé son auteur à la rédiger. Ce dialogue politique a déplu à certains, et c'est en partie à cause de cela que d'autres licences de logiciels libres ont été rédigées. Cependant, la GPL a été rédigée avec le concours de professeurs de droit, elle est donc bien mieux écrite que la plupart des autres licences de son genre. Je vous encourage vivement à employer la GPL, ou sa variante pour bibliothèques la LGPL, si vous le pouvez. Si vous choisissez une autre licence, ou écrivez la vôtre propre, vérifiez que les raisons qui vous poussent à cela sont valables. Il faut bien expliquer à ceux qui écrivent leurs propres licences que ce n'est pas là une décision à prendre à la légère. Les complications inattendues qu'une licence mal écrite peut entraîner peuvent créer, pendant plusieurs décennies, un fardeau pour les utilisateurs du logiciel.

Le texte de la GPL n'est pas lui-même placé sous la GPL. Cette licence est simple : quiconque a le droit de copier et de distribuer des copies exactes du document de cette licence, mais pas de le modifier. Il faut remarquer que le texte des licences open source n'est pas en général pas open source car, à l'évidence, une licence que tout un chacun pourrait modifier n'offrirait qu'une protection limitée.

Les dispositions de la GPL satisfont la définition de l'open source. La GPL n'exige aucune des clauses autorisées par le paragraphe4 de la définition de l'open source, « Intégrité du code source de l'auteur ».

La GPL ne vous autorise pas à rendre des modifications secrètes. Vos modifications doivent elles aussi être distribuées selon les termes de la GPL. Ainsi, l'auteur d'un programme sous GPL recevra probablement des améliorations proposées par d'autres, y compris des sociétés commerciales qui modifient son logiciel pour leurs besoins propres.

La GPL n'autorise l'incorporation d'un programme GPL au sein d'un programme propriétaire. La définition de « programme propriétaire » utilisée par cette licence est : « tout programme dont la licence vous donne moins de droits que ne vous en donne la GPL. »

La GPL contient quelques échappatoires qui lui permettent d'être utilisée dans des programmes qui ne sont pas entièrement open source. On peut lier les bibliothèques de logiciels qui sont distribuées avec le compilateur ou le système d'exploitation avec des logiciels protégés par la GPL ; il en résulte un programme partiellement libre. Le détenteur du copyright (qui est en général l'auteur du programme) est la personne qui place la GPL sur son programme et qui dispose du droit de violer sa propre licence. C'est ce que faisaient les auteurs de KDE pour distribuer leurs programmes avec Qt avant que la société Troll Tech ne place la bibliothèque Qt sous une licence open source. Cependant, ce droit ne s'étend pas aux tiers qui redistribuent le programmeils doivent suivre tous les termes de la licence, même ceux que le détenteur du copyright viole, c'est pourquoi il est problématique de redistribuer un programme GPL s'il contient Qt. Les développeurs de KDE semblent avoir résolu ce problème en appliquant la LGPL, et non la GPL, à leurs logiciels.

La rhétorique politique de la GPL déplaît à certaines personnes, dont une partie ont choisi pour leurs logiciels une licence moins appropriée pour la seule raison qu'ils fuient les idées de RichardStallman et ne souhaitent pas les voir répétées dans leurs propres paquetages de logiciels.

12.6.2 La licence publique générale de GNU pour les bibliothèques

La LGPL est dérivée de la GPL et mise au point pour les bibliothèques de logiciels. À la différence de ce qui se produit avec la GPL, on peut incorporer un programme sous LGPL dans un programme propriétaire. La bibliothèque du langage C fournie avec les systèmes GNU/Linux est un exemple de logiciel sous LGPLon peut l'utiliser pour construire des programmes propriétaires, sans quoi Linux ne serait utile qu'aux auteurs de logiciels libres.

On peut à tout moment convertir une instance d'un programme LGPL en programme GPL. Mais on ne peut ensuite plus reconvertir cette instance, ni tout ce qui en dérive, en programme sous LGPL.

Les autres dispositions de la LGPL sont semblables à celles de la GPLen fait, cette licence inclut la GPL en y faisant référence.

12.6.3 Les licences X, BSD, et Apache

La licence X et celles qui lui sont prochesles licences BSD et Apache sont très différentes de la GPL et de la LGPL. Elles vous autorisent à tout faire avec les logiciels qu'elles protègent. Cela, parce que les logiciels que les licences X et BSD couvraient à l'origine était financés par des bourses du gouvernement des États-Unis d'Amérique. Puisque les citoyens des États-Unis d'Amérique avaient déjà payé pour ces logiciels une fois, avec leurs impôts, ils ont reçu la permission d'utiliser ces logiciels comme bon leur semblait.

La permission la plus importante, qu'on ne trouve pas dans la GPL, est qu'on peut rendre secrètes des modifications apportées à un programme sous licence X. En d'autres termes, vous pouvez récupérer le code source d'un programme protégé par la licence X, le modifier, et vendre ensuite des versions binaires de ce programme sans en distribuer le code source correspondant, et sans appliquer la licence X à ces modifications. Cela n'en reste pas moins de l'open source, car la définition de l'open source n'exige pas que les modifications soient sous la licence originale.

Nombreux furent les autres développeurs qui ont adopté la licence X et ses variantes, parmi lesquels on remarquera on particulier la BSD (distribution du système de Berkeley) et le projet de serveur web Apache. Une spécificité ennuyeuse de la licence BSD est qu'une de ses dispositions exige qu'on mentionne (généralement dans une note de bas de page) que le logiciel a été développé à l'université de Californie à chaque fois qu'on fait la publicité d'un aspect d'un programme couvert par la licence BSD. Garder la trace des logiciels sous licence BSD dans un ensemble aussi vaste qu'une distribution GNU /Linux, et penser à mentionner l'université de Californie à chaque fois qu'on mentionne l'un de ces programmes dans une publicité, sont des tâches trop lourdes pour ceux qui sont dans les affaires. Au moment où j'écris ces lignes, la distribution Debian GNU/Linux compte plus de 2500 paquetages logiciels, et même si seule une fraction d'entre eux était sous la licence BSD, toute publicité pour un système

GNU/Linux comme la distribution Debian impliquerait des pages et des pages de notes ! Cependant, la licence du Consortium X ne comporte pas cette clause relative à la publicité. Si vous envisagez d'utiliser une licence de la famille BSD, choisissez plutôt la licence X.

12.6.4 La licence Artistique

Même si cette licence, à l'origine, a été développée pour Perl, elle a depuis été employée pour d'autres logiciels. Je pense que c'est une licence rédigée sans soin, en ce sens qu'elle pose des exigences et qu'elle donne ensuite des échappatoires pour les contourner. C'est peut-être la raison pour laquelle presque tous les logiciels couverts par la licence Artistic le sont désormais par deux licences, offrant le choix entre l'Artistic et la GPL.

La section 5 de la licence Artistic interdit la vente du logiciel, mais autorise la vente d'une distribution contenant plusieurs logiciels. Ainsi, il suffit d'empaqueter un programme couvert par la licence Artistic dans un ensemble contenant également un « Hello world ! » de cinq lignes écrit en C pour avoir le droit de vendre ce paquet. C'est cette spécificité de la licence Artistic qui était la seule cause de l'échappatoire « composant d'une distribution d'un ensemble... » du premier paragraphe de la définition de l'open source. L'utilisation de la licence Artistic déclinant, nous pensons ôter cette précision. Cela ferait de la licence Artistic une licence non open source. Ce n'est pas là une décision à prendre à la légère, et nous y penserons et en débattrons probablement pendant plus d'un an avant de la prendre.

La licence Artistic vous oblige à libérer les modifications que vous pouvez être amené à faire, tout en vous laissant une échappatoire (dans la section7) qui vous autorise à garder secrètes des modifications, ou même à placer des portions d'un programme sous licence Artistic dans le domaine public !

12.6.5 La licence publique de Netscape et la licence publique de Mozilla

La NPL a été développée par la société Netscape quand elle a rendu son produit, Netscape Navigator, open source. En réalité, la version open source s'appelle Mozilla ; les gens de Netscape réservent la marque Navigator à leur propre produit. EricRaymond et moi intervînmes en tant que conseillers non rémunérés tout au long du développement de cette licence. J'ai tenté, sans succès, de persuader la société Netscape d'utiliser la GPL, et suite à leur refus, je les ai aidés à composer une licence qui remplirait les conditions de la définition de l'open source.

La NPL a ceci de particulier qu'elle renferme des privilèges particuliers, dont seul jouit la société Netscape. Elle lui accorde le droit de placer des modifications apportées à leur logiciel par un tiers sous une autre licence. Les gens de Netscape peuvent rendre ces modifications secrètes, les améliorer, et refuser de communiquer le résultat. Cette clause était nécessaire car quand la société Netscape a décidé de placer son produit principal sous une licence open source, elle était sous contrat avec d'autres sociétés qui ont exigé d'obtenir le logiciel Navigator sous une licence non open source.

La société Netscape a créé la MPL, ou licence publique de Mozilla, pour régler ce problème. La MPL ressemble beaucoup à la NPL, mais ne contient pas

la clause qui autorise Netscape à placer des modifications extérieures sous une autre licence.

La NPL et la MPL vous autorisent à rendre certaines modifications secrètes.

De nombreuses sociétés ont adopté une variante de la MPL pour leurs propres programmes. C'est regrettable, car la NPL a été créée en fonction de la situation commerciale particulière dans laquelle la société Netscape se trouvait alors, et elle n'est pas forcément appropriée à d'autres usages. Il vaut mieux qu'elle demeure la licence de Netscape et de Mozilla, et que les autres utilisent la GPL ou la licence X.

12.7 Choisir une licence

N'écrivez pas une nouvelle licence si vous pouvez employer l'une de celles qui sont listées ici. La propagation de licences nombreuses et incompatibles cause du tort au logiciel open source car on ne peut pas utiliser des fragments d'un programme au sein d'un autre programme si les deux licences qui les protègent sont incompatibles.

Abstenez-vous d'utiliser la licence Artistic à moins que vous ne prévoyiez de l'étudier attentivement et d'en ôter les échappatoires. Puis, prenez quelques décisions :

1. Souhaitez-vous ou non que l'on puisse rendre des modifications secrètes ? Si vous souhaitez pouvoir obtenir, de la part de ceux qui les ont apportées, le code source des modifications, utilisez une licence qui l'exige. La GPL et la LGPL sont de bons candidats. Si cela ne vous ennuie pas qu'on puisse modifier votre programme sans vous en rendre compte, utilisez les licences X ou Apache.
2. Souhaitez-vous que quelqu'un puisse fusionner votre programme avec son propre logiciel, propriétaire ? Si c'est le cas, utilisez la LGPL, qui autorise explicitement cela sans permettre à autrui d'apporter des modifications à votre propre code sans les publier, ou utilisez les licences Apache ou X, qui autorisent le fait de rendre des modifications secrètes.
3. Souhaitez-vous qu'on puisse vendre des versions de votre programme sous licence commerciale, non open source ? Si c'est le cas, distribuez votre programme sous les termes de deux licences. Je recommande la GPL en tant que licence open source ; vous pourrez trouver une licence commerciale appropriée dans des livres tels que Copyright Your Software (protégez vos logiciels grâce au copyright), édité par Nolo Press.
4. Souhaitez-vous que tous ceux qui utilisent votre programme acquittent une somme pour bénéficier de ce privilège ? Si c'est le cas, alors peut-être que l'open source n'est pas pour vous. Si souhaitez que certains utilisateurs vous donnent de l'argent, c'est toujours possible en publiant votre programme sous une licence open source. La plupart des auteurs de logiciels open source considèrent que leurs programmes sont des contributions au bien public, et se fichent de recevoir ou non de l'argent pour ces derniers.

- Mélange : le programme peut être mélangé avec du logiciel non libre
- Modifications secrètes : les modifications peuvent être rendues secrètes et non communiquées à l'auteur

TAB. 12.1 – Comparaison des pratiques des différentes licences

Licence	Mélange	Modifications secrètes	Nouvelle licence	Privilèges
GPL	Non	Non	Non	Non
LGPL	Oui	Non	Non	Non
BSD	Oui	Oui	Non	Non
NPL	Oui	Oui	Non	Oui
MPL	Oui	Oui	Non	Non
Domaine public	Oui	Oui	Oui	Non

- Nouvelle licence : tout un chacun peut placer le programme sous une nouvelle licence
- Privilèges : la licence renferme des privilèges particuliers pour le détenteur du copyright, qui peuvent s'appliquer à vos modifications

12.8 L'avenir

Au moment où cet essai était sur le point de partir sous presse, la société IBM a rejoint le monde de l'open source, et la communauté du capital-risque découvre elle aussi l'open source. Les sociétés Intel et Netscape ont investi dans la société Red Hat, qui distribue Linux. La société VA Research, qui intègre des serveurs Linux sur des stations de travail, a annoncé qu'elle disposait d'un investisseur extérieur. La société Sendmail Inc., créée pour commercialiser sendmail, programme de distribution du courrier électronique qu'on trouve partout, a annoncé qu'elle disposait de six millions de dollars américains de fonds. Le programme de courrier électronique, Postfix, de la société IBM, est proposé selon les termes d'une licence open source, et un autre programme de la société IBM, le compilateur Jikes pour le langage Java, est proposé selon les termes d'une licence qui tente, à l'heure où j'écris ces lignes, de remplir les clauses de la définition de l'open source, sans y parvenir tout à fait. La société IBM semble vouloir modifier la licence de Jikes pour que ce programme soit vraiment open source, et rassemble les commentaires de la communauté en ce moment même.

Deux rapports internes à la société Microsoft, plus connus sous le nom de documents de Halloween, ont été portés à la connaissance de tous. Ces rapports documentent clairement le fait que la société Microsoft se sent menacée par le mouvement de l'open source et par Linux, et que la société Microsoft les attaquera tous deux pour protéger ses marchés. Il est évident que nous vivons une époque intéressante. Je pense que vous verrons la société Microsoft utiliser ses deux stratégies principales : placer les interfaces sous copyright, et protéger ses programmes à l'aide de brevets. Elle étendra les protocoles réseau, en y incluant des spécificités propres à Microsoft, qui ne seront pas disponibles pour le logiciel libre. Elle explorera, et d'autres sociétés avec elle, de nouveaux axes de recherche en informatique et pourra tout breveter avant que la communauté du logiciel libre ne puisse employer ces nouvelles techniques, et elle nous en maintiendra à distance respectueuse en exigeant le versement de sommes conséquentes pour avoir le droit d'utiliser ces brevets. J'ai écrit un essai pour le magazine sur le web Linux World expliquant comment combattre les ennemis de l'open source

sur le front des brevets.

La bonne nouvelle, c'est que la société Microsoft a peur ! Dans le deuxième document Halloween, un employé de cette société s'extasie sur le sentiment qu'il a éprouvé en constatant qu'il pouvait facilement modifier des portions du système Linux pour lui faire remplir ses besoins, et que cela était bien plus facile dans le cas de Linux que ce ne l'était pour un employé de Microsoft de modifier MS-WindowsNT !

Les coups de bélier provenant de l'intérieur sont les plus dangereux. Je pense que vous verrons également de plus en plus de tentatives de diluer la définition de l'open source pour lui faire inclure des produits partiellement libres, comme on l'a vu dans le cas de la bibliothèque Qt de KDE avant que la société Troll Tech n'ait été éclairée et n'ait fourni une licence open source. La société Microsoft, comme d'autres, pourraient nous causer du tort en produisant énormément de logiciels, suffisamment libres pour attirer les utilisateurs, sans pour autant proposer toutes les libertés de l'open source. On peut concevoir qu'ils pourraient tuer le développement de certaines catégories de logiciels open source en fournissant des solutions « assez bonnes » et « presque assez libres ». Cependant, la forte réaction à l'encontre du projet KDE, avant que la bibliothèque Qt ne soit pleinement open source, laisse présager que de telles tentatives de la part de la société Microsoft et de ses semblables échoueront.

Jusqu'à présent, nous avons échappé aux chevaux de Troie. Mais supposons que quelqu'un qui ne nous aime pas propose un logiciel renfermant un cheval de Troie, une manière cachée de casser la sécurité d'un système GNU/Linux. Supposons ensuite que cette personne patiente jusqu'au moment où son logiciel sera bien répandu avant de rendre publique la faille et sa vulnérabilité. Le grand public aura alors vu que notre système, l'open source, peut nous laisser plus démunis face à ce type d'attaque que ne le sont les systèmes fermés de la société Microsoft, ce qui risque de porter un coup à sa confiance dans les logiciels open source. On pourra rétorquer que la société Microsoft a son compte de problèmes de sécurité, et qu'elle n'a pas besoin pour les insérer de gens extérieurs mal intentionnés, et que le modèle de l'open source, exposant le code source des programmes à la vue de tous, facilite la découverte de tels bogues. Tout bogue de ce type se produisant sur Linux sera corrigé le lendemain de sa publication, alors qu'un tel problème sous MS-Windows peut demeurer indécelé ou non corrigé pendant des années. Ils nous faut cependant améliorer notre défense face aux chevaux de Troie. Bien identifier ceux qui proposent des logiciels et des modifications représente notre meilleure défense, et cela nous permet de poursuivre en justice ceux qui diffusent des chevaux de Troie. Quand je dirigeais la distribution Debian GNU/Linux, nous avions mis en place un système pour identifier de manière sûre tous ceux qui maintenaient des logiciels et pour qu'ils prennent part dans un réseau de cryptographie à clé publique qui nous permettrait de vérifier de qui provenait chaque logiciel. Ce type de système a pris de l'ampleur pour finalement inclure tous les développeurs de logiciels open source.

Il nous faut encore faire des améliorations gigantesques avant que Linux puisse être employé par l'utilisateur moyen. Nous manquons clairement d'interfaces graphiques, et ce sont les projets KDE et GNOME qui traitent ce déficit. Notre prochain but sera alors de faciliter l'administration du système : le logiciel linuxconf a beau régler partiellement cela, il est loin de proposer à l'utilisateur débutant un outil exhaustif d'administration système. Si le système COAS de la société Caldera tient ses promesses, il pourrait servir de base à une solution

complète d'administration. Malheureusement, la société Caldera n'a pas pu allouer des ressources suffisantes pour que le développement du projet COAS soit mené à terme, et d'autres participants ont abandonné le projet, démotivés par une progression trop lente, voire inexistante.

La pléthore de distributeurs de GNU/Linux semble subir un écrémage, dont la société Red Hat sortira comme vainqueur annoncé, talonnée par la société Caldera. La société Red Hat a montré jusqu'à présent un solide attachement au concept de l'open source, mais l'avènement d'un nouveau président et les rumeurs d'une offre publique d'achat (OPA) pourraient être le signe d'un affaiblissement de cet attachement, surtout si des concurrents comme la société Caldera, qui ont manifesté bien moins de marques d'intérêt dans l'open source, entament les marchés de Red Hat. Si l'attachement des distributions commerciales de GNU/Linux au modèle de l'open source devait par trop s'affaiblir, on verrait vraisemblablement se mettre en place des efforts pour les remplacer par des distributions entièrement open source, telles que Debian GNU/Linux, plus ciblées vers les marchés commerciaux que cela n'a été le cas de la distribution Debian.

Malgré tous ces défis, je prédis que l'open source remportera la victoire. Le système GNU/Linux est devenu le système de tests des étudiants en informatique, et ces derniers introduiront ces systèmes libres dans leur entreprise, quand ils auront obtenu leur diplôme. Les laboratoires de recherche ont adopté le modèle de l'open source car le partage de l'information est essentiel dans la méthode scientifique, et l'open source permet de partager facilement le logiciel. Les entreprises adoptent elles aussi le modèle de l'open source car il permet à des groupes de sociétés de collaborer pour résoudre un problème sans craindre une poursuite en justice pour situation de monopole et à cause du gain apporté à leurs logiciels par les contributions libres des programmeurs du grand public. Certaines grandes sociétés ont adopté l'open source en tant que stratégie pour combattre la société Microsoft et s'assurer qu'aucun autre Microsoft ne dominera jamais plus l'industrie de l'informatique. Mais c'est dans le passé qu'il faut chercher l'indication la plus fiable de l'avenir de l'open source : en à peine quelques années, on est passé de presque rien à un solide fonds de logiciels qui résolvent de nombreux problèmes différents, et on est en passe d'atteindre le million d'utilisateurs. Nous n'avons aucune raison de réduire la voilure maintenant.

Chapitre 13

Matériel, logiciel et infoware

Je discutais récemment avec quelques amis qui ne possèdent pas d'ordinateur. Ils pensaient qu'en avoir un leur ouvrirait les portes du site web Amazon.com où ils souhaitent acheter des livres et des CD. Non pas accéder à « l'Internet », ni même au « le Web ». Mais utiliser Amazon.com tel était leur unique désir.

Une application de ce genre est dite « gagnante », car elle incite quelqu'un à acheter un ordinateur.

Ce qui est intéressant, c'est qu'une application gagnante n'est plus une application de bureautique, ou même un logiciel de backoffice d'entreprise, mais un simple site web. Lorsque l'on commence à penser aux sites comme à des applications, on réalise bientôt qu'ils constituent une race entièrement nouvelle, que l'on pourrait appeler « applications de l'information » ou peut-être même infoware [1] ».

Les applications sont employées pour informatiser des tâches que l'ancien modèle informatique ne prenait pas en charge. Quelques années plus tôt, si vous aviez voulu effectuer une recherche dans une base de données d'un million d'ouvrages, vous auriez discuté avec un libraire connaissant les arcanes de la syntaxe des outils de recherche informatisée. Pour un livre, vous vous seriez rendu dans une librairie, vous auriez cherché dans une sélection relativement limitée. Aujourd'hui, des dizaines de milliers de personnes sans aucune formation particulière trouvent et achètent tous les jours des livres en ligne à partir de cette base de données contenant des millions d'enregistrements.

Le secret, c'est que les ordinateurs ont facilité le dialogue entre les gens. Les applications web utilisent des interfaces en langue courante et des images, et non de petits codes spécialisés qui n'ont de signification que pour ceux qui maîtrisent cette discipline.

Les logiciels traditionnels embarquent de petits morceaux d'information dans un tas de logiciels ; les infowares posent quant à eux quelques logiciels sur de vastes gisements d'informations. Les « actions » dans un produit infoware sont généralement simples : faire un choix, acheter ou vendre, saisir quelques données et obtenir un résultat personnalisé.

Des scripts attachés à un lien hypertexte réalisent souvent ces actions en utilisant une interface spécifique appelée CGI (Common Gateway Interface). Un CGI défini un moyen pour un serveur web d'appeler un programme externe et de publier la sortie de ce programme sous la forme d'une page web. Les programmes CGI peuvent être simplement de petits scripts qui réalisent un calcul simple, ou qui exploitent un serveur de données. Mais même lorsqu'un moteur logiciel lourd se trouve derrière un site, l'interface utilisateur n'est pas composé de logiciels traditionnels. Elle consiste en un ensemble de pages web (qui ont été créées par un auteur, un éditeur ou un concepteur graphique plutôt que par un programmeur).

Les interfaces informatives sont typiquement dynamiques. La présentation des livres d'Amazon, par exemple, est pilotée par les résultats des ventes mis à jour toutes les heures. Les clients peuvent ajouter des commentaires et des estimations en ligne qui deviennent un élément clé dans le processus de décision des acheteurs. Un site conçu pour aider quelqu'un à acheter ou vendre des produits en ligne doit non seulement de présenter ses prix à jour mais aussi fournir des nouvelles pertinentes, des informations commerciales internes, des recommandations d'analystes et peut-être même un forum de discussion pour les utilisateurs.

1. ce qui correspondrait en français à un néologisme comme « informiciel

Cette interface d'information consiste en un riche mélange de documents réalisés de façon artisanale, de données générées par des programmes, et de liens vers des serveurs d'applications spécialisées (comme le courrier électronique, la discussion informelle en ligne ou bien la téléconférence).

Lorsqu'il s'agit d'assurer des taches répétitives, les interfaces d'informations ne sont pas aussi efficaces que celles des logiciels, mais elles sont bien meilleures que celles des outils que vous employez rarement ou différemment à chaque fois. Elles sont en particulier adéquates lorsqu'il faut choisir parmi les éléments d'information présentés. Pendant que vous achetez un livre ou un CD ; chez Amazon, ou un produit chez E*Trade, l'achat réel est une partie assez triviale de l'interaction. C'est la qualité de l'information procurée qui vous aide à prendre la décision, elle est le cœur de l'application avec laquelle vous interagissez.

L'ère du Web est en train de transformer tout le paradigme informatique qui n'a jamais été aussi clair pour moi que depuis 1994, lorsque Microsoft se convertit au Web, et que j'ai partagé la scène (via satellite) avec le vice-président de cette société, Craig Mundie, lors d'un événement au Japon organisé par NTT. Mundie montrait l'interface prévue pour le serveur Microsoft « Tiger », qui était supposé diffuser de la vidéo à la demande. L'interface émulait Microsoft-Windows, avec des menus déroulants qui répondaient à une télécommande virtuelle.

Il était assez évident pour ceux qui sont impliqués dans le Web, que l' interface adéquate pour la vidéo à la demande, lorsqu'elle viendra et si elle vient, sera une interface de type Web. Ironie du sort, Microsoft possédait l'interface parfaite pour la vidéo à la demande : sa propre encyclopédie du cinéma sur CDROM, intitulée Cinemania. Quoi de mieux que de choisir le film souhaité en cherchant sa catégorie, lisant quelques critiques et visionnant quelques passages, puis de cliquer sur un lien hypertexte afin de lancer la projection ? Cinemania dispose de tout cela sauf de la dernière étape. Les capacités réelles des produits d'information hypertextuels ne deviennent patentes qu'après leur connexion à des réseaux. L'information n'est pas une fin en soi, mais une interface grâce à laquelle l'utilisateur de contrôler l'espace d'une application beaucoup trop complexe d'un logiciel traditionnel. (Amazon le sait bien : leur achat d'une base de données de films sur l'Internet, une collection de critiques de spectateurs et d'autres informations sur les films, les placera en première position non seulement pour vendre de cassettes vidéo en ligne mais comme un futur point de passage obligé pour les futurs services de vidéo à la demande).

Les interfaces d'informations sont particulièrement appropriées pour des applications d'aide à la décision, mais elles sont également utiles pour des tâches que l'on ne réalise qu'une seule fois. Dans un sens, l'utilisation « d'automates-magiciens » pour une installation de logiciels est un exemple de cette tendance.

Certaines applications d'informations utilisent une interface plus simple, dont le mode d'interaction avec l'utilisateur est plus proche des programmes informatiques classiques. Mon exemple favori est une application qui était virtuellement impensable il a seulement quelques années : obtenir des cartes et des consignes de route. Un site de cartographie comme maps.yahoo.com vous permet de saisir deux adresses et vous renvoie une carte et un ensemble de consignes à suivre pour aller de l'une à l'autre.

Quel rapport avec les logiciels libres ?

Il existe une réponse évidente : la plupart des techniques qui permettent au Web de fonctionner sont libres.

Les caractéristiques de l'Internet, les protocoles réseau TCPIP ;, et tous les

éléments clés de l'infrastructure, comme le système de résolution de noms (DNS Domain Name System) ont été développés selon le modèle Open Source. Il est facile de démontrer que le programme libre BIND (Berkeley Internet Name Daemon), qui assure ce service est l'application chargée d'un rôle critique la plus répandue. Même si la plupart des navigations web sont réalisées avec des produits propriétaires (Navigator de Netscape ou Internet Explorer de Microsoft), ils sont tous deux des bâtis autour du Web, donc de spécifications ouvertes édictées par Tim Berners-Lee . Selon les données du serveur web Netcraft, qui réalise un comptage automatique, plus de 50% des serveurs web visibles seraient réalisés avec le serveur web libre Apache. La majorité des contenus dynamiques est engendrée par des langages de scripts libres comme Perl, Python, et Tcl.

Mais cette réponse, si elle évidente, n'est que partielle. Après tout, c'est le Web qui est à la base des applications d'informations réseau du futur et non pas les techniques propriétaires.

À l'heure actuelle, Microsoft négocie le virage en prenant conscience de la puissance du multimédia en ligne. En 1994, lorsque le Web démarra, Microsoft, avec ses produits sur CDROM comme Encarta, son encyclopédie en ligne, et Cinemania ;, sa référence de films, s'y dirigea en mettant en ligne des documents avec des liens hypertexte forts de riches capacités multimédia. Microsoft réalisa même qu'il était important de fournir des sources d'information sur les réseaux.

Cette vision du réseau selon Microsoft (Microsoft Network) ne souffrait que d'un seul problème : la barre d'admission était haut placée. Les éditeurs devaient utiliser des outils propriétaires Microsoft, être approuvés par Microsoft, et payer pour avoir le droit de « jouer ». Au contraire, n'importe qui pouvait créer un site web. Les logiciels dont il avait besoin étaient gratuits. Les spécifications pour créer des documents et du contenu dynamique étaient simples, ouvertes et clairement documentées.

Ce qui était peut-être plus important encore, c'est que les techniques et l'éthique de l'Internet légitimaient la copie d'éléments entre les sites web. Les pages HTML (HyperText Markup Language), qui étaient utilisées pour implanter diverses caractéristiques d'un site web, pouvaient facilement être sauvegardées et imitées. Les scripts CGI eux-mêmes, utilisés pour créer un contenu dynamique étaient disponibles, et on pouvait s'en inspirer. Même si les langages de programmation comme C sont plus rapides, Perl est devenu le langage dominant pour les CGI car il est plus facilement accessible. Perl est suffisamment puissant pour écrire de grosses applications, tout en permettant à des amateurs d'écrire de petits scripts pour effectuer des tâches spécialisées. Encore plus important, Perl n'étant pas un langage compilé, les scripts qui sont utilisés dans des pages web peuvent être examinés, copiés et modifiés par des utilisateurs. De plus, des archives de scripts utiles en Perl sont réalisées et échangées librement entre les développeurs de sites web. La facilité du clonage de sites web construits avec la combinaison HTML+CGI+Perl rend pour la première fois possible la création d'applications réseau puissantes par des non-informaticiens.

À cet égard, il est intéressant de faire remarquer que les principaux efforts de l'industrie du logiciel pour améliorer l'interface web des techniques à contenu actif, comme les applets Java et les ActiveX de Microsoft, ont échoué car ils visaient les programmeurs professionnels et ne pouvaient pas être facilement copiés par les amateurs qui construisaient le Web. Les vendeurs voient le Web en termes de logiciels, ils ne comprennent pas que ce dernier est en train de changer, non seulement les applications qui sont en cours d'élaboration, mais

également les outils dont les créateurs ont besoin.

Les analystes de l'industrie ont prédit depuis plusieurs années que Perl et les CGI seraient éclipsés par de nouvelles techniques. Mais même à l'heure actuelle, alors que les plus grands sites web emploient des équipes importantes de programmeurs professionnels, et que de nouvelles techniques comme les ASP (Active Server Pages) de Microsoft et les servlets Java de Sun supplantent les CGI pour des raisons de performances, Perl continue d'accroître sa popularité. Avec les autres langages de scripts libres comme Python et Tcl, il reste au cœur des sites web, petits et grands, car les applications infowares sont fondamentalement différentes des applications logicielles, qui nécessitent d'autres outils.

Si vous étudiez un gros site comme Yahoo !, vous verrez dans les coulisses une armée d'administrateurs et de programmeurs qui reconstruisent continuellement le produit. Le contenu dynamique n'est pas engendré automatiquement, il est souvent façonné à la main, typiquement en utilisant une batterie de scripts écrits rapidement et salement.

« Chez Yahoo !, nous ne créons pas de contenu. Nous le regroupons et le mettons en forme. » explique Jeffrey Friedl , auteur du livre Mastering Regular Expressions et programmeur à temps complet chez Yahoo !. « Nous nous nourrissons de milliers de sources, chacune à son propre format. Nous faisons des quantités de traitements pour purifier et trouver ce qu'il faut mettre sur Yahoo ! ». Par exemple, pour lier les nouvelles au bandeau déroulant de quotes.yahoo.com, Mr Friedl a eu besoin d'écrire un programme de « reconnaissance de noms » qui soit capable de chercher plus de 15000 noms de sociétés. La capacité d'analyse de texte de Perl et la puissance des expressions rationnelles ont rendu cela possible.

Perl est également un composant central dans l'administration système des infrastructures utilisées pour conserver à un site son caractère vivant et actualisé. Un grand nombre de scripts Perl arpentent continuellement les serveurs Yahoo ! et leurs liens sur les sites externes et alertent le personnel lorsqu'une URL ne renvoie pas le résultat escompté. Le plus connu de ces « arpenteurs » est référencé comme la « faucheuse macabre (Grim Reaper) ». Si une connexion automatisée à une URL échoue plus d'un certain nombre de fois, le document en question est enlevé du répertoire de Yahoo !.

Amazon est également un grand utilisateur de Perl. L'environnement d'édition démontre le puissance de Perl à rassembler des outils informatiques disparates ; c'est un « langage d'assemblage » par excellence. Un utilisateur crée un nouveau document avec un formulaire qui appelle un programme Perl, ce dernier engendre un document partiellement au format SGML, puis lance soit Microsoft Word soit GNU Emacs (au choix de l'utilisateur), mais intègre aussi CVS (Concurrent Versioning System) et les outils maison d'Amazon.com. Les classes SGML d'Amazon sont utilisées pour présenter différentes sections sur le Web à titre d'exemple (en HTML, avec ou sans graphiques, à partir de la même base). Un analyseur syntaxique transforme le SGML en HTML pour approbation, avant que les auteurs ne valident leurs modifications.

Perl est qualifié de « chatterton de l'Internet », et, tel le « chatterton », il est utilisé dans les cas les plus inattendus. Comme un ensemble de films mis bout à bout avec du ruban adhésif, un site web est souvent ouvert et fermé en un jour, et nécessite des outils légers et rapides, mais une solution efficace.

Les vains efforts de Microsoft pour retourner de l'Infoware ; au logiciel avec les ActiveX sont empêtrés sur le plan conceptuel, en constante évolution dans

l'industrie informatique. Lorsqu'un segment particulier de marché est mature, les acteurs existants ont tout intérêt à ce que les choses restent en place. Adopter ce qui est réellement nouveau ne leur facilite pas la tâche et permet presque de réclamer que de nouveaux acteurs (« les barbares », pour citer Philippe Kahn [2]) créent de nouveaux marchés.

La suprématie acquise par Microsoft sur IBM grâce à la position dominante qu'il détient sur l'industrie informatique est un bon exemple des plus récents développements de la situation. IBM a laissé le marché à Microsoft car il n'avait pas perçu que le changement de pouvoir n'était pas limité au passage de « l'ordinateur format armoire à glace » au modèle de bureau, mais également du matériel propriétaire au produit de consommation, et du matériel au logiciel.

De la même manière, en dépit de ses tentatives pour entrer dans les affaires liées à l'information, Microsoft ne réalise pas, et, si cela se trouve, ne peut réaliser, que le logiciel tel qu'il le conçoit ne sera plus la principale source de création de valeur du secteur.

Du temps de la suprématie d'IBM, le matériel était roi, et la barre pour entrer dans les affaires informatiques était haut placée. La plupart des logiciels étaient créés par des constructeurs de matériels ou par leurs éditeurs satellites.

La disponibilité du PC comme plate-forme de consommation courante (ainsi que le développement des systèmes ouverts tels qu'Unix ;) a radicalement changé les règles. Soudainement, la barre d'entrée s'est retrouvée à un niveau bas et les entrepreneurs tels que Mitch Kapor, de Lotus, et Bill Gates, de Microsoft, décollèrent.

Si vous explorez l'histoire récente du Web, vous verrez un modèle semblable. Le monopole de Microsoft sur la bureautique a monté la barre extrêmement haut pour quiconque voudrait le concurrencer. Qui plus est, les applications informatiques sont devenues de plus en plus complexes avec Microsoft qui dresse délibérément des barrières pour entraver la concurrence. Ce n'est plus guère possible pour un simple programmeur dans un garage (ou dans une mansarde) d'avoir un quelconque impact.

C'est peut-être la caractéristique la plus importante du logiciel libre : il abaisse les barrières d'entrée sur le marché du logiciel. Vous pouvez librement essayer un nouveau produit et mieux encore, vous pouvez librement créer votre propre version. Le code source est à la disposition d'une quantité importante de vos pairs à des fins de révision. Si l'un d'eux n'en aime pas un détail, il peut le retirer, le corriger ou le ré-implanter. Les corrections étant souvent proposées à la communauté, les erreurs peuvent être rapidement corrigées.

Les développeurs (au moins au début) n'essaient pas de faire concurrence au monde des affaires, mais s'astreignent néanmoins à résoudre des problèmes réels, et de ce fait il y a une place pour l'expérimentation dans un environnement moins exacerbé. Comme je l'ai souvent entendu dire, les logiciels Open Source « vous laissent vous gratter là où ça vous démange ». Grâce au paradigme du développement distribué, et avec les nouvelles fonctionnalités ajoutées par d'autres utilisateurs, les programmes libres évoluent au fur et à mesure de leur conception.

En effet, l'évolution du marché opère mieux à l'instar de la nature en ne s'encombrant pas de barrières marketing, sortes de prothèses qui facilitent la survie des inadaptés. L'évolution engendre non pas un seul vainqueur, mais

2. le fondateur de Borland

toute une gamme de gagnants.

C'est précisément l'idiosyncrasie de nombreux programmes libres qui fait leur force. Cela renforce encore les raisons du succès de Perl.

Au départ, Larry Wall a conçu Perl afin d'automatiser des tâches d'administration système auxquelles il était confronté. Après avoir mis son logiciel à disposition du réseau, il lui trouva de plus en plus d'applications, et le langage s'étoffa souvent dans des directions inattendues.

Perl a été décrit comme un langage « jetable » à cause de ses caractéristiques, qui semblent chaotiques aux concepteurs de langages plus « carrés ». Mais le chaos révèle souvent une structure riche. Il semble nécessaire pour modéliser ce qui est par essence complexe. Les langages humains sont complexes car ils modélisent la réalité. Comme l'écrit Mr Wall dans l'essai qu'il a rédigé pour le présent livre :

> L'anglais est utile parce qu'il est désordonné. Comme il est désordonné, il s'applique bien à l'espace des problèmes, lui aussi désordonné, que l'on appelle réalité. De façon similaire, Perl a été construit pour être désordonné (mais de la façon la plus agréable possible).

Le développement libre est une approche incroyablement efficace lorsqu'il s'agit de faire travailler les développeurs sur les fonctionnalités importantes. Un nouveau logiciel est développé en restant au plus près des demande des utilisateurs, sans distorsion causée par le marketing, sans être issu de décisions d'achat formulées par de hauts responsables. La méthode de développement ascendante est idéale pour résoudre les problèmes issus de la base.

Des entrepreneurs comme Jerry Yang et David Filo ont réussi grâce aux logiciels libres, au cœur du Web, et au paradigme de développement libre, plus simple. Ce n'est pas par accident que Yahoo !, le site web le plus important et le plus couronné de succès, soit construit autour de logiciels libres, librement disponibles : le système d'exploitation FreeBSD, Apache, et Perl.

Comme il y a quelques temps, le passage à une grande échelle sera l'étape suivante. Bob Young , de RedHat, le leader des distributions Linux, l'a noté : il ne se propose pas de détrôner Microsoft sur le marché des systèmes d'exploitation mais plutôt de réduire la valeur globale (montants dépensés cumulés) de ce secteur.

Le fait est que le logiciel libre n'a pas besoin de battre Microsoft à son propre jeu. Cependant, il change la nature du jeu.

Malgré toute la capitalisation boursière du marché, les prestataires d'applications informatives comme Amazon et Yahoo ! demeurent petits devant Microsoft. Ce qui se dessine est clair. C'est dans les Infowares, et non dans le logiciel, qu'on trouvera des pistes pour les interactions homme-machine, et des occasions d'informatiser des tâches qui ne l'avaient pas été jusqu'alors.

Alors qu'émergent les « applications tueuses », le rôle du logiciel consistera de plus en plus à rendre les Infowares accessibles. Fournir des serveur web, l'accès aux bases de données, des serveurs d'applications et des langages de programmation comme Java restera profitable tant que ces nouveaux produits entreront eux-mêmes dans le nouveau modèle plutôt que d'essayer de le supplanter. Notons que lors du passage de l'industrie informatique centrée sur le matériel à l'industrie centrée sur le logiciel, le matériel n'a pas disparu. IBM continue d'être une entreprise florissante (cependant, la plupart de ses pairs ont

régressé voire échoué). Mais d'autres acteurs ont émergé, adaptés aux nouvelles règles : Dell, Compaq, et surtout Intel.

Intel a bien compris que cela ne leur offrait pas une occasion de gagner la guerre des systèmes informatiques mais plutôt d'être une arme au service des combattants.

Le véritable défi pour le logiciel libre n'est pas de savoir s'il remplacera Microsoft dans la domination de l'ordinateur de bureau, mais plutôt d'engendrer un modèle métier qui l'aidera à devenir le « IntelInside ; » de la nouvelle génération d'applications informatiques.

Sinon, les pionniers du logiciel libre seront mis à l'écart comme Digital Research le fut de la course aux systèmes d'exploitation des PC, par quelqu'un qui avait bien compris où se trouvaient les occasions de profit.

Quelle que soit l'évolution de tout cela les logiciels libres ont déjà ouvert une nouvelle voie. Comme les pionniers des premiers micro-ordinateurs (en logiciel et matériel) ont créé la première étape de l'industrie d'aujourd'hui, les logiciels libres ont créé une étape dans un drame qui ne fait que commencer, et qui va conduire à une re-définition du paysage de l'industrie informatique pour les cinq ou dix ans à venir.

Chapitre 14

Libérons les sources — L'histoire de Mozilla

14.1 Introduction

Le 23 janvier 1998, Netscape publia 2 annonces. Voici comment CNet rapporta la première : « Dans un élan sans précédent, Netscape Communication fera cadeau de son navigateur Navigator, confirmant les rumeurs des dernières semaines passées. »

Voici le compte-rendu de la seconde : « Ils donneront aussi le code source pour la prochaine génération de sa suite Communicator. »

La décision de donner son navigateur ne surprit personne, mais la livraison du code source étonna toute l'industrie. Cela occupa les gros titres des journaux dans le monde entier, et même la communauté du logiciel libre fut surprise de cet élan. Jamais auparavant un grand éditeur de logiciels n'avait fourni ainsi son code. Qu'allait devenir Netscape ?

Nous avions décidé de changer les règles du jeu, mais ce n'était pas la première fois. Connu depuis toujours pour penser différemment, Netscape prenait ainsi l'engagement de construire un Internet meilleur, de le hausser à un autre niveau. Lorsque Netscape commença la distribution sans restriction de versions de son navigateur sur l'Internet en 1994, certains déclarèrent : « C'est complètement fou ! ». Quand Netscape annonce « Code Source libre », ils dit la même chose.

La période de discussion amenant à l'annonce de l'ouverture des sources se déroula à vitesse grand V. Alors qu'il avait fallu des mois de délibérations pour décider s'il fallait fournir ou non le logiciel gratuitement, 24 heures suffirent à une forte proportion des responsables pour accepter de libérer les sources.

Aussi rapide et surprenante que fut l'annonce pour le personnel et les observateurs, elle reflétait plusieurs courants de pensée convergents. Les dirigeants de Netscape discutèrent d'un livre blanc rédigé par Frank Hecker qui y exprimait une opinion de première importance car il prônait la libéralisation de leurs sources. Frank y citait l'article d'Eric Raymond, « La Cathédrale et le Bazar », en en parlant avec les gens de toute l'organisation, que ce soit les ingénieurs, le marketing ou le management. Dans un opus de 20 pages qui fut très diffusé, il plaidait la cause dont le bataillon des convaincus s'étoffait régulièrement.

Quand Netscape rendit pour la première fois Navigator téléchargeable sans restriction sur l'Internet de nombreux observateurs crurent voir une atteinte au bon sens classique de l'industrie du logiciel, et doutèrent que nous puissions gagner de l'argent « en faisant cadeau du logiciel ». Aujourd'hui, naturellement, cette stratégie est rétrospectivement perçue comme une innovation couronnée de succès, un facteur clé dans le développement rapide de Netscape ; et rare sont aujourd'hui les sociétés qui ne copient pas notre stratégie d'une façon ou d'une autre. Cela soulève diverses questions, par exemple : que se passerait-il si nous reproduisions le même scénario, cette fois-ci avec le code source ?

Les ingénieurs se prononçaient de façon similaire. De nombreux employés de Netscape avaient l'habitude de travailler avec des outils libres. Et depuis que le code de Communicator était si intimement mêlé avec Java et HTML, beaucoup admettaient une vérité émergente : libérer le source pose à présent moins de problème. La nature de Java invite à une perception plus ouverte de la distribution des sources. Il est multi-plate-forme et peut être compilé en fichiers classes, exécutables indépendants de la machine car chaque binaire repose sur une machine virtuelle, donc les programmeurs peuvent décompiler l'exécutable

et en étudier le code source. La commande « Voir le source » du navigateur, elle, assure la popularité de HTML d'une façon semblable. Plutôt qu'essayer de bloquer cela, beaucoup pensaient que Netscape devait le faciliter, l'encourager, et en tirer autant que faire se peut bénéfice.

Les différentes écoles de pensée fusionnèrent avec une soudaineté inattendue. Durant les réunions, la stupéfaction laissait en quelques minutes place à l'approbation. La plupart des discussions passèrent rapidement de « devons-nous ? » à « quand ? ». La plupart des acteurs clés croyaient que nous devions agir rapidement, annoncer une date ferme puis s'y tenir. En janvier, Netscape publia une promesse sur le Net : les sources de Communicator seraient livrées durant le premier trimestre de 1998. Netscape tint cette promesse avec le plus grand sérieux, et le « Project Source 331 », nom donné au fruit du travail des équipes uvrant afin de publier le 31 mars 1998, vit le jour.

Puis la réalité s'imposa.

14.2 Rendre cela possible

Le corps du code source de Communicator fut appelé « Mozilla », nom crée par Jamie Zawinsky et la société durant le développement du Navigator. L'équipe travaillait frénétiquement pour créer une « bête » beaucoup plus puissante que Mosaic, et ce nom devint celui du code officiel de Navigator. Plus tard le gros dinosaure vert devint une blague interne, puis une mascotte pour la société, et enfin un symbole pour le public. Le nom est à présent devenu le terme générique se référant aux navigateurs Web libres dérivant du code source de Netscape Navigator. On s'employait à « libérer le Lézard ».

Une impressionnante quantité de tâches restait à accomplir afin de préparer la première publication du code. Les problèmes apparaissaient peu à peu et furent collectés et classés dans des catégories. Le trimestre suivant fut consacré à leur résolution, à la cadence infernale que Netscape connaissait bien.

La mise à disposition des modules de tierces parties inclus dans le navigateur posa l'un des plus gros problèmes. Les trois-quarts du code de Communicator en relevaient et nous devions donc contacter tous leurs propriétaires. Des équipes d'ingénieurs et d'évangélistes furent organisées pour rencontrer et convaincre chaque société de rejoindre Netscape sur la route des Sources Libres. Tous avaient entendu parler de l'annonce de Netscape et devaient choisir : leur code pouvait être supprimé ou remplacé, livré en binaire (conservé dans son état compilé) ou bien livré sous forme de code source avec Communicator. Pour compliquer les choses, de nombreux contrats étaient uniques et portaient sur différentes échéances. Aucun scénario ne pouvait s'appliquer à tous les cas.

Nous souhaitions respecter la date de publication annoncée et cela nécessitait des choix parfois difficiles. Ce fut sûrement le cas quand vint la question de la participation des développeurs des tierces parties. La règle était qu'ils devaient nous rejoindre avant le 24 février, ou leur élément serait écarté des sources. Ce type de date butoir n'est pas difficile à établir lorsque l'échéance est encore lointaine mais devint critique quand elle fut proche. Une partie du code dut être supprimée.

Les sources de la machine virtuelle Java, langage propriétaire, ne devaient pas être publiés. Trois ingénieurs se virent confier une « Java-ectomie » grâce à laquelle le navigateur pourrait être compilé puis s'exécuter sans Java. La modifi-

cation du code général, très intimement mêlé à Java, exigea beaucoup de travail. Nous nous proposions de consacrer les deux dernières semaines aux tests. Les ingénieurs devaient démêler tout le code Java du navigateur avant le 15 mars, échéance très proche.

Le nettoyage du code fut un projet énorme. Beaucoup doutèrent très tôt de la date avancée mais les cerveaux commençaient à bouillir lors de réunions, des stratégies prirent forme, les roues commencèrent à tourner. L'équipe de production laissa de côté son travail (la plupart de ses membres développaient la génération suivante du navigateur) et tous s'attaquèrent immédiatement au projet. Il fallait non seulement décider du sort (inclusion ou suppression) de tout module issu d'une tierce partie mais aussi mettre au propre les commentaires du code. La responsabilité de chaque module fut assignée à une équipe et ils commencèrent à récurer.

Une des grandes innovations qui se produisit tôt fut d'utiliser le système de rapport de bogues offert par notre Intranet en tant que gestionnaire de tâches. « Bugsplat » était le surnom de Scopus, un programme de rapport de bogues à interface en HTML. Ce système de gestion des flux de documents nous épaula au mieux. Il tenait à jour la liste des nouvelles tâches décrites grâce à un simple formulaire HTML et gérait les priorités des tâches et les personnes compétentes comme autant de bogues ainsi pris en charge. Les listes de diffusion électroniques correspondantes naissaient à mesure. Sitôt la tâche (ou le bogue) résolu, toutes les listes de diffusion et les priorités disparaissaient. Les ingénieurs mesuraient l'avancement de leurs propres modules et surveillaient le déroulement du projet grâce à notre Intranet.

L'équipe d'ingénieurs peina aussi afin de supprimer les modules de chiffrement. Le gouvernement insistait non seulement pour que tout ces codes disparaissent mais encore pour que toute fonction qui y faisait appel soit réécrite. Le seul travail d'une des équipes fut de garder un contact constant avec la NSA et de gérer les problèmes de conformité.

14.3 Mise au point de la licence

Alors que le Grand Nettoyage du Code battait son plein, un autre projet devait décider de la licence de diffusion à adopter. Il nous fallait pour cela répondre tout d'abord à une importante question : l'une des licences existantes était-elle adéquate ? Personne ne souhaitait rédiger de nouvelle licence, mais tout le monde réalisa qu'il serait nécessaire de satisfaire les tierces parties et d'adopter une démarche fédérant les entreprises. Aucun logiciel propriétaire existant n'avait jamais été fourni ainsi.

Un groupe de leaders de la communauté Open Source incluant Linus Torvalds, Eric Raymond et Tim O'Reilly fut invité à visiter le campus de Mountain View. Ils participèrent à des débats avec des groupes de dirigeants, de juristes et de programmeurs afin d'exposer leurs vues et rencontrèrent de petits groupes pour discuter de problèmes spécifiques. Ils consacrèrent une grande partie de leur temps à discuter avec l'équipe juridique de Netscape des mérites et défauts des licences existantes. Ces conseillers agirent aussi comme un groupe prolifique en idées.

Une équipe conseillée par l'équipe juridique de Netscape examina les licences existantes afin de déterminer si l'une d'entre elles pouvait s'appliquer à Mozilla.

Ils commencèrent par examiner la licence publique générale GNU destinée aux bibliothèques (la LGPL) et la licence BSD, et nous avons longuement analysé leurs avantages et inconvénients respectifs. Le source de base de Netscape présentait des caractéristiques propres, bien distinctes de celles des codes auxquels elles avaient été déjà appliquées. Les accords de licence ad hoc qui régissaient de nombreux composants issus de tierces parties utilisés posaient l'un des problèmes les plus épineux. La licence devait créer un environnement où ces contributeurs et d'autres nouveaux développeurs commerciaux pourraient participer au développement de Mozilla tout en préservant leurs intérêts.

La licence BSD, plus permissive, qui requiert seulement que le détenteur légal mentionné dans tout source dérivé, fut jugée insuffisante dans le cas de Mozilla. Elle aurait exposé les développeurs à constater que leurs modifications ne seraient pas reversées à la communauté. Ce point unique posait un énorme problème puisqu'il était crucial pour la viabilité à long terme des travaux de développement open source.

Les contraintes de la GPL la rendaient incompatible avec notre projet. La GPL est « contagieuse » car doit aussi protéger tout code combiné par compilation avec un code sous GPL. Cette exigence la rend inadéquate dans le cas de logiciels commerciaux. La GPL, par exemple, concernerait aussi les composants des parties tierces compilés dans les versions de Communicator, ce dont Netscape ne saurait décider, puisque nous n'en disposons pas. Et Netscape lui-même utilise une partie du code de Communicator dans d'autres produits (par exemple des serveurs). Puisque Netscape n'avait aucun plan immédiat pour fournir leur code source, l'effet viral de la GPL sur ces produits présentait un problème pour Netscape autant que pour les autres entreprises. La LGPL, une modification de la GPL plus ouverte et moins restrictive, était plus proche des besoins de Netscape pour un développement commercial, mais elle limitait les modalités de commercialisation, tout comme la GPL.

Après un mois frénétique de recherche, de discussion et de rencontres avec des experts et défenseurs du logiciel libre, et au milieu des spéculations du public, l'équipe décida qu'une nouvelle licence pourrait seule correspondre à cette situation sans précédent. La Licence Publique de Netscape (NPL), compromis entre la promotion du développement des Sources Libres par des entreprises commerciales et la protection des développeurs concernés, naquit ainsi. Le processus de création d'une nouvelle génération de licence Open Source exigea plus d'un mois.

Dans un autre élan extraordinaire le brouillon de sa toute première version fut présenté au public. Le 5 mars, une version de la NPL fut postée dans un nouveau forum de nouvelles Usenet appelé netscape.public.mozilla.license, assortie d'une demande invitant tout lecteur intéressé à la commenter. Encouragements et moqueries fusèrent. La section de la licence qui garantissait à Netscape le droit d'utiliser le code couvert par la NPL dans d'autres produits sans devoir placer ces derniers sous NPL faisait l'objet de la plupart des critiques. Cela permit aussi à Netscape de publier des versions révisées de la NPL, et de manière très controversée, de re-licencier du code sous NPL aux parties tierces sous des termes différents de ceux de cette licence. Certaines des personnes faisant des commentaires en retour, allaient si loin qu'elles suggéraient que ce seul fait rendrait la NPL inacceptable pour la communauté des développeurs Open Source.

jwz (Jamie Zawinsky) publia le 11 mars dans netscape.public.mozilla.license un état des travaux dont voici un extrait :

Premièrement, MERCI pour la somme incroyable de commentaires pertinents expédiés ! Ce fut incroyablement utile, et soyez assurés que les opinions exprimées sont très scrupuleusement prises en considération. La semaine prochaine, vous pouvez vous attendre à voir une section 5 très retravaillée. Je ne pourrai probablement pas trop la commenter (car ne souhaite pas donner de faux espoirs), mais ce que la plupart d'entre vous détestent a été clairement circonscrit.

Le 21 mars, la modification fut postée. C'était sans précédent. La réaction fut incroyable : « Je leur ai dit que c'était affreux et ils m'ont écouté ! Je n'arrive pas à y croire ! ». Le public réalisa qu'il s'agissait bien d'un projet open source, malgré son lieu de naissance surprenant. Les utilisateurs des forums Usenet, au lieu de commenter comme à l'ordinaire les résultats, guidèrent le processus de mise au point. Le ton changea dans les discussions, et la bonne volonté s'y montra davantage.

La critique de la version bêta de la NPL émise par la communauté avait renvoyé l'équipe de licence à sa table de travail. Ils cherchèrent une solution qui permettrait à Netscape de soutenir les objectifs des développeurs de logiciels libres tout en servant ses objectifs commerciaux. Le résultat fut une deuxième version de la NPL appelée Licence Publique Mozilla (MozPL). Elles sont identiques à ceci près que la NPL comporte des amendements laissant à Netscape des droits additionnels.

Tout le code réalisé pour le 31 mars 1998 fut publié sous la NPL, et toutes les modifications de ce code le seront sous NPL. Un nouveau code développé peut être publié sous la MozPL ou sous tout autre licence compatible avec elle. Des changements dans des fichiers contenus dans le code source sont considérés comme des modifications et placés sous NPL. Pour répondre à de nombreux doutes exprimés sur le réseau, les nouveaux fichiers qui ne contiennent pas de code original ou de code modifié ultérieurement ne sont pas considérés comme des modifications et ne sont donc pas couverts par la NPL. Le code résultant peut être placé sous n'importe quelle licence compatible. La GPL n'est pas compatible avec la Licence Publique Netscape ou avec la Licence Publique Mozilla car semble incompatible avec toute autre licence puisque elle interdit toute addition de restrictions ou de plus amples permissions. Tout code développé pour fonctionner avec du logiciel GPL doit à son tour être protégé par la GPL. Un autre point mineur est que la GPL insiste pour que, quand vous distribuez du code sous ses termes, elle doit être complète et entière. La NPL ne comporte pas cette condition.

Les discussions dans les forums Usenet soulignèrent un important problème : des développeurs souhaitaient voir Netscape distinguer les corrections de bogue et le nouveau code. Affirmer : « Je réalise une correction de bogue, une petite modification de votre programme, » est tout-à-fait différent de « J'ajoute une nouvelle fonction à votre programme ». Ces assertions provoquent des réactions différentes. La plupart des gens acceptent d'offrir une correction de bogue, et la récompense d'une contribution réside dans son don même. Mais il n'en va pas de même pour du code additionnel. Un développeur productif ne souhaite pas voir n'importe qui utiliser son travail afin de gagner de l'argent.

La NPL et la MozPL furent conçues pour encourager le développement ouvert sur le code de base de Mozilla, mais dès le début un autre objectif se dessina. Netscape souhaitait être la première grande entreprise à ouvrir son code propriétaire parce qu'elle voulait mieux entretenir l'intérêt d'autres grandes so-

ciétés pour l'environnement open source. Il lui semblait primordial de créer une atmosphère propice, pousser les organisations rentables à adopter ce modèle et à participer au mouvement. L'infrastructure légale de la plupart des licences libres n'offrait que des obstacles à la coopération de sociétés. C'est pourquoi la licence de Mozilla constituait à elle seule un projet à part entière.

Nous espérions, en offrant le code source des futures versions, inviter toute la communauté de l'Internet à créer un nouveau mode d'innovation dans le marché du navigateur. L'idée selon laquelle de talentueux programmeurs à travers le monde travailleraient sur notre code, apportant au navigateur leur énergie créative, motivait tout le monde à aller de l'avant même si la progression s'avérait difficile.

14.4 mozilla.org

Les personnes qui avaient participé à des projets open source auparavant se rendaient compte que le code devait disposer d'un site de référence. Jamie enregistra la nuit même un nouveau nom de domaine et rédigea la charte des projets de développement distribués grâce à mozilla.org, qui naquit ainsi.

Les projets open source couronnés de succès suivaient tous un modèle, pas nécessairement à dessein, par lequel une personne ou un groupe assure la coordination. Les personnes travaillent sur l'aspect du code qui leur plaît, selon leurs propres envies. À la fin de la journée chacun dispose d'un module fonctionnant un peu mieux. Mais lors de la publication d'une nouvelle version du logiciel ils doivent adapter leurs modifications car le morceau de source du logiciel principal les concernant a peut-être évolué.

Les développeurs souhaitent donc voir leurs modifications intégrées dans la distribution principale. Un amas de code source et modifié sans méthode par de intervenants poussera tôt ou tard quelqu'un à se dresser et dire « Je pourrais très bien collecter une masse de modifications et constituer ainsi une nouvelle version ». Un autre contributeur pourra dès lors penser « Je ne sais pas à qui remettre ma modification, alors je vais le donner à ce type. Il semble en faire bon usage. » Le temps passe, et cette personne devient le mainteneur.

Pour ce projet open source, on mit la charrue avant les bufs. Mozilla.org fut conçu et réalisé afin de remplir le rôle de mainteneur dès le début. Puisque cette mission serait d'une manière ou d'une autre prise en charge, nous décidâmes de créer l'infrastructure pour devenir les gestionnaires du travail.

En quelques mois mozilla.org commença à devenir une organisation, trouvant des fonds et des machines, gérant des listes de diffusion électroniques, et tissant les appuis nécessaires afin que tout fonctionne. La mission consistait simplement à préserver une organisation adéquate et fonctionnelle. Le dépôt central devait être opérationnel dès que les sources seraient publiées sous peine de voir après quelques mois un tiers endosser cette responsabilité. Netscape, comme chacun le sait, n'est pas du genre à regarder les autres faire à sa place.

Offrir le code source signifiait que Netscape collaborerait avec les utilisateurs de l'Internet. Il nous fallait impérativement bien distinguer le groupe de développement des produits clients Netscape et celui de mozilla.org. mozilla.org devait agir en tant que coordinateur de toutes les développeurs actifs tandis que l'objectif du développement produit était de coordonner les produits Netscape réalisés à partir du code de Mozilla. Puisque les deux groupes travaillent sur

le même produit, les intérêts pouvaient se chevaucher. Mais le groupe derrière mozilla.org reconnut qu'il serait désastreux pour son image sur le Net que quelqu'un se tourne vers l'organisation et affirme « Ces gens n'ont que les intérêts de Netscape en tête et veulent seulement vendre leurs produits. ». Ceci signifierait que mozilla.org n'est pas le mainteneur adéquat. La séparation devait être réelle et les activistes de l'Internet le savaient.

14.5 En coulisses

Que se passe-t-il lorsque l'auteur d'une modification nous interpelle : « Eh, mozilla.org ! Prenez ce code s'il vous plaît ! ». L'un des rôles les plus importants de mozilla.org est de définir les critères d'acceptation d'une contribution. Nous devons faire face à de nombreux problèmes. Tout d'abord vient l'appréciation du mérite. Le code est-il bon ? Deuxièmement, est-il sous une licence compatible avec la NPL ? Nous décidâmes de ne pas accepter les contributions non couvertes par une licence compatible avec la NPL. Sinon elles devraient être ventilées dans des répertoires bien distincts, séparés par des murailles, et cela impliquerait de grandes manuvres d'ordre juridique. Le risque d'erreur augmenterait énormément.

Puisque Mozilla est une base de code très modulaire chacun de ses composants majeurs, par exemple la bibliothèque de gestion des images ou l'analyseur XML, a un « propriétaire » désigné. Il connaît parfaitement le code et fait office d'arbitre lorsque l'on décide d'y inclure ou non une contribution.

De nombreux propriétaires de modules sont des ingénieurs de Netscape mais d'autres sont des volontaires externes. Quand un propriétaire de module effectue des changements (par exemple, ajoutant une API à la bibliothèque de gestion des images), il les expédie à mozilla.org qui les inclura dans les distributions. Si des différents apparaissent entre un contributeur et le propriétaire du module, mozilla.org joue le rôle d'arbitre en établissant l'appel final — toujours conscient que s'il arrête de jouer selon les règles de l'art il sera ignoré et quelqu'un d'autre endossera sa mission.

Mozilla.org devait faire face à la diversité des développeurs. Les méthodes utilisées pour travailler sur le code en interne devaient s'accommoder du Web et rester accessibles pour toutes les plates-formes, dans toutes les zones horaires. Ce fût fait avec le « contrôle d'arborescence » assuré par les outils Bonsai et Tinderbox.

L'outil « Bonsai » honore des requêtes portant sur le contenu d'une archive. Comme aux guichets d'une banque, vous pouvez grâce à lui « déposer dans la chambre forte » du code développé, et voir quels « dépôts » ont été effectués par d'autres participants. En arrière-plan il engendre constamment le code, vérifiant l'arborescence du source et lève un « drapeau rouge » en cas de problème, interdisant les dépôts jusqu'à ce que le problème soit identifié. Il engendre sur simple demande des historiques et la liste des problèmes pris en charge durant une période donnée. Bonsai, tout d'abord utilisé par les développeurs maison de Netscape, fut imposé aux activistes du monde entier uvrant, qui y accédaient sur mozilla.org grâce à n'importe quel arpenteur Web.

Plus de dix développeurs travaillant simultanément sans outils adéquats mènent à la catastrophe. Un programme nommé « Tinderbox » gère ce travail d'équipe afin d'éviter cela en menant une sorte d'enquête dans l'arborescence des

sources. Grâce à lui on connaît toujours l'identité de l'auteur d'un quelconque tronçon de code (il exploite pour cela Bonsai), quelles versions du logiciel fonctionnent ou non (et pourquoi !), et l'état des fichiers utilisables afin d'identifier les sources des défauts.

14.6 Premier avril 1998

Une semaine et demie avant la fin de mars 1998 nous avions l'impression que le temps filait de plus en plus vite vers la date butoir. Tous souhaitaient organiser une célébration de la distribution du code, mais rien n'avait été organisé. Une fête de cette nature, couplée à la publication prévue, deviendrait un événement sensationnel qui inviterait le public à pénétrer librement dans le monde de Netscape.

Lors une réunion Jamie exposa son plan de louer une boite de nuit à San Francisco, d'inviter du monde et de l'annoncer sur le Net. « Vous pensez inviter des personnes non salariées par Netscape à la fête ? Mais nous n'avons jamais fait cela auparavant ! ». Après une pause la réaction fut, comme durant tout ce projet, « Pourquoi pas ? ».

Cette fête ne sera pas oubliée de sitôt. Jamie loua « The Sound Factory », l'une des plus grosses boîtes de nuit de San Francisco, pour la nuit du 1er avril. Des DJ (notamment Brian Behlendorf, l'un des fondateurs d'Apache) firent cadeau de centaines de T-shirts mozilla.org, de logiciels, et d'objets de NetObjects, Macromedia, Digital, Be, Wired, et de diverses sociétés non américaines.

Une longue file d'attente s'étendait déjà devant les portes de cette la « Mozilla Dot Party » lorsqu'elles s'ouvrirent à 20 heures. Une heure et demie plus tard, l'endroit était rempli à la limite tolérée par les pompiers (deux mille personnes), et la file d'attente faisait le tour du pâté de maisons. Les gens étaient accueillis par vingtaines à l'heure où d'autres s'en allaient, et à la fin de la nuit, plus de 3500 personnes avaient passé les portes, y compris des gourous du logiciel libre comme Brewster Kahle (fondateur de WAIS) et Eric Raymond. Des centaines d'autres fans, à travers le monde, synchronisèrent leurs montres et portent un toast à la santé de Mozilla. Les fêtards virtuels incluaient plus d'une centaine de personnes au château de Waag à Amsterdam, Pays-Bas, et des groupes individuels variés en Norvège, à Montréal, Canada, en Pennsylvanie, Caroline du Nord, Colorado, Alabama et dans le Wisconsin.

À l'intérieur, trois écrans de projection déroulaient le code à la vitesse de soixante lignes par seconde. Avec ce débit, la fête aurait pu durer plus de sept heures afin d'afficher ainsi le million et demi de lignes du source de Mozilla. Eric Raymond, durant le second des deux concerts du Kofy Brown Band (parmi lesquels figure un ingénieur de Netscape), qui était venu ici de Philadelphie pour l'occasion, sauta sur la scène et surprit tout le monde avec un solo de flûte. À la fin de la nuit, une douzaine de CD de l'édition Signature du code source de Mozilla (CD signés et numérotés avant la nuit par l'équipe de d'intégration Netscape et des membres de mozilla.org) furent jetés dans la foule et saisi par quelques heureux privilégiés. Le lézard était libre !

Chapitre 15

La revanche des hackers

15.1 La revanche des hackers

J'ai écrit la première version de ce document intitulé « Une brève histoire des hackers » en 1996, pour le web. J'avais été fasciné par la culture des hackers depuis de longues années, bien avant d'avoir édité la première édition de « The New Hacker's Dictionary », en 1990. À la fin de l'année 1993, nombreux étaient ceux (et j'en faisais partie) qui en étaient venus à me considérer comme l'historien de la tribu des hackers, et leur ethnographe dépêché sur place. Ce rôle me plaisait bien.

Je n'imaginais alors pas que mon anthropologie amateur se révélerait être un catalyseur significatif qui provoquerait des changements. Je fus plus surpris que quiconque de le constater. Mais les conséquences de cette surprise résonnent encore de nos jours sur la culture des hackers et les mondes des affaires et des nouvelles techniques.

Dans cet essai je récapitulerai, de mon point de vue, les événements qui ont immédiatement précédé la « coup de canon de retentissement mondial », de janvier 1998, de la révolution de l'OpenSource. Je présenterai mes réflexions sur la distance remarquable qui a été parcourue depuis. Enfin, j'offrirai quelques projections tentantes sur l'avenir.

15.2 Au-delà de la loi de Brooks

J'ai rencontré Linux pour la première fois à la fin de l'année 1993, à travers une distribution sur CD-ROM du pionnier Yggdrasil. À cette époque, j'avais déjà été impliqué dans la culture des hackers depuis plus de quinze ans. Mes premières expériences remontaient à l'ARPAnet primitif de la fin des années 1970 ; j'ai même été brièvement touriste sur les machines ITS. J'avais déjà écrit du logiciel libre et je l'avais publié sur l'Usenet avant que la Free Software Foundation ne voie le jour en 1984, et je fus l'un des premiers contributeurs à la FSF. Je venais de publier la deuxième édition de The New Hacker's Dictionary. Je pensais très bien connaître la culture des hackers et ses limitations.

La rencontre avec Linux fut un choc. J'avais beau avoir baroudé de nombreuses années dans la culture des hackers, je traînais toujours l'idée reçue que des hackers amateurs, même très doués, ne pourraient jamais rassembler suffisamment de ressources ou de talent pour produire un système d'exploitation multi-tâches utilisable. Les développeurs du Hurd, après tout, avaient ostensiblement échoué sur ce point pendant une décennie.

Mais là où ils ont échoué, LinusTorvalds et sa communauté ont réussi. Et ils ne se sont pas contentés de ne remplir que les exigences minimales en matière de stabilité et d'interfaces fonctionnant à la Unix. Oh non. Ils ont explosé ces critères par leur exubérance et leur flair, en fournissant des centaines de millions d'octets de programmes, de documents, et d'autres ressources. Des suites complètes d'outils pour l'Internet, des logiciels de publication de qualité professionnelle, la possibilité d'utiliser le mode graphique, des éditeurs, des jeux tout cela, et bien plus encore.

Observer cette débauche de codes merveilleux étalés sous mes yeux fut une expérience bien plus puissante que de me contenter de savoir, d'un point de vue uniquement intellectuel, que toutes les portions existaient probablement déjà quelque part. C'est comme si je m'étais baladé au milieu de piles de pièces de

rechange dépareillées pendant des années pour me retrouver face aux même pièces, assemblées sous la forme d'une Ferrari rouge et rutilante, portière ouverte, les clés se balançant sur le contact, et le moteur ronronnant des promesses de puissance...

La tradition des hackers, que j'avais observée pendant vingt ans, semblait soudain prendre vie d'une nouvelle et vibrante manière. Dans un certain sens, je faisais déjà partie de cette communauté, car plusieurs de mes projets de logiciel libre avaient été ajoutés à la mêlée. Mais je voulais y pénétrer plus profondément, car chacune des merveilles que j'observais approfondissait aussi mon étonnement. C'était trop beau !

Les coutumes du génie logiciel sont inféodées à la loi de Brooks, qui prédit que lorsque le nombre N de programmeurs augmente, le travail accompli augmente en proportion mais que la complexité et la vulnérabilité aux bogues augmente en N^2. N^2 est le nombre de chemins de communication (et d'interfaces de code potentielles) séparant les bases de code des développeurs.

La loi de Brooks prédit qu'un projet comptant des milliers de contributeurs devrait être un capharnaüm floconneux et instable. D'une certaine manière, la communauté de Linux a vaincu l'effet N^2 en produisant un système d'exploitation d'une qualité exceptionnelle. J'étais décidé à comprendre la manière dont ils avaient procédé.

Il m'a fallu trois ans de participation et d'observation de près pour développer une théorie, et une année encore pour la mettre en pratique. Je me suis alors assis à mon bureau et j'ai rédigé « La cathédrale et le bazar », alias CatB, pour expliquer ce que j'avais vu.

15.3 Des mimiques et des mythes

Ce que je voyais autour de moi, c'était une communauté qui avait mis au point la méthode de développement logiciel la plus efficace de tous les temps, sans même s'en rendre compte ! C'est-à-dire qu'une pratique efficace s'était mise en place sous la forme d'un ensemble de coutumes, transmises par l'imitation et l'exemple, sans la théorie ou le langage qui permette d'expliquer pourquoi la pratique fonctionnait. En y repensant, l'absence de cette théorie et de ce langage nous a gêné de deux manières. D'abord, on ne pouvait pas mettre en place une réflexion systématique sur la manière d'améliorer nos propres méthodes. Ensuite, on ne pouvait pas expliquer ni vendre la méthode à d'autres. À l'époque, seul le premier effet retenait mon attention. Ma seule intention, en écrivant ce papier, était de donner à la culture des hackers le langage approprié, qu'elle utiliserait de manière interne, pour s'expliquer à elle-même. C'est ainsi que j'ai couché sur le papier ce que j'avais vu, en lui donnant l'allure d'une narration et en utilisant des métaphores vivantes et appropriées pour décrire la logique qu'on pouvait deviner derrière ces coutumes. « CatB » ne contient pas vraiment de théorie fondamentale. Je n'ai inventé aucune des méthodes qu'il décrit. Ce qui est nouveau, dans ce papier, ce ne sont pas les faits, mais les métaphores et la narration une histoire simple et puissante qui encourageait le lecteur à voir les faits sous un jour nouveau. J'essayais d'appliquer le génie mimétique aux mythes fondateurs de la culture des hackers. J'ai d'abord soumis le papier complet au Linux Kongress de mai 1997, en Bavière. L'intense attention et les applaudissements nourris qu'il a suscités de la part d'un public ne contenant que

quelques personnes dont l'anglais était la langue maternelle semblaient confirmer que j'étais sur quelque chose d'important. Mais il se trouve que le hasard, qui m'a placé aux côtés de TimO'Reilly lors du banquet du jeudi soir, a ébranlé un ensemble de conséquences plus important. J'admirais le style des éditions O'Reilly depuis longtemps, je brûlais donc de rencontrer Tim O'Reilly depuis plusieurs années. Notre conversation embrassa de nombreux thèmes (et en particulier notre intérêt commun pour la science-fiction classique) ce qui provoqua mon invitation à la conférence de Perl, donnée par Tim plus tard la même année, afin d'y présenter « CatB ». Une fois encore, le papier fut bien reçu il obtint, en réalité, des acclamations et l'auditoire se leva pour lui rendre un vibrant hommage. Le courrier électronique que je recevais m'avait appris que depuis la Bavière, le bouche à oreille avait rempli son office sur l'Internet, à propos de « CatB », plus rapidement qu'un feu de brousse. Nombreux étaient ceux qui dans l'assistance l'avaient déjà lu, et mon discours fut moins une révélation pour eux qu'une occasion de célébrer le nouveau langage, et la prise de conscience qui l'accompagnait. La salle ne s'est pas tant levée pour me rendre hommage que pour célébrer la culture des hackers elle-même et en cela, elle avait bigrement raison. Je ne le savais pas encore, mais mon expérience en ingénierie mimétique était sur le point de bouter un feu bien plus important. Certains de ceux qui découvrirent mon discours ce jour-là travaillaient pour la société Netscape Communications, Inc, qui avait des problèmes. Netscape, pionnier des nouvelles techniques de l'Internet, extravagant de WallStreet, était sur la liste noire de la société Microsoft. Cette dernière craignait à juste titre que les normes pour le web, incarnées par le navigateur de la société Netscape, n'érodent le monopole lucratif dont le géant de Redmond disposait alors sur la plate-forme des compatibles PC. Tout le poids de ses milliards, ainsi que ses tactiques inavouables, qui lui vaudraient quelque temps plus tard d'être poursuivi dans le cadre de la loi antitrust, étaient alors déployés pour anéantir le navigateur de la société Netscape. Pour la société Netscape, le problème était moins le revenu associé à son navigateur (qui ne représentait qu'une faible proportion de ses recettes) que de maintenir une zone de sécurité pour les affaires associées à leur serveur, bien plus lucratives. Si le navigateur Microsoft Internet Explorer se trouvait en position dominante sur le marché, cette dernière pourrait corrompre les protocoles du web en les éloignant des normes ouvertes pour les transformer en canaux propriétaires que seuls ses propres serveurs pourraient proposer. À l'intérieur de la société Netscape, le débat battait son plein sur la manière de contrer la menace. Une option proposée dès le début fut de libérer le code source du navigateur mais cette position était difficile à tenir en l'absence de bonnes raisons de croire que cela empêcherait la domination du logiciel Internet Explorer. Je ne le savais pas encore alors, mais « CatB » fut un avocat déterminant de cette position. Au cours de l'hiver 1997, alors que je travaillais sur mon prochain article, tout était prêt pour que la société Netscape abandonne les règles du jeu du commerce habituel et offre à ma tribu une occasion sans précédent.

15.4 La route de Mountain View

Le 22janvier 1998, la société Netscape annonçait qu'elle publierait le code source de son client pour le web sur l'Internet. Je n'appris cette nouvelle que le lendemain, pour apprendre peu après que Jim Barksdale, le PDG, avait présenté

mon travail aux journalistes qui couvraient l'événement nationalement comme une « inspiration fondamentale » pour sa décision.

C'est cet événement que les commentateurs de la presse professionnelle en informatique appelleraient plus tard « coup de canon de retentissement mondial » et M.Barksdale avait fait de moi son Thomas Paine[1], que je le veuille ou non. Pour la première fois dans l'histoire de la culture des hackers, l'une des entreprises préférées du groupe Fortune500 avait parié son avenir sur la croyance que les hackers avaient raison. Et, plus spécifiquement, que mon analyse de la culture des hackers était correcte.

C'est là un choc bien difficile à encaisser. Je n'avais pas vraiment été surpris que « CatB » modifie l'image que la culture des hackers avait d'elle-même ; c'est ce que j'avais cherché à faire, après tout. Mais j'ai été soufflé (et c'est peu dire) par les nouvelles du succès qu'il rencontrait à l'extérieur. C'est pourquoi j'ai réfléchi intensément pendant quelques heures, après avoir appris la nouvelle. J'ai réfléchi à l'état de Linux et de la communauté des hackers. J'ai réfléchi à celui de Netscape. Puis me suis demandé si j'étais assez costaud pour faire le prochain pas.

Il n'était pas difficile de conclure qu'aider Netscape à réussir son pari venait d'acquérir le statut de priorité fondamentale pour la culture des hackers, et par conséquent, pour moi personnellement. Si ce pari échouait, les hackers subiraient probablement l'opprobre de cet échec de plein fouet. Nous serions discrédités pendant encore dix ans. Et ce seraient dix ans de trop.

À cette époque je faisais partie de la culture des hackers et je vivais ses différentes phases depuis vingt ans. Pendant vingt ans j'avais observé des idées brillantes, des débuts prometteurs, et des techniques supérieures se faire invariablement écraser par une mercatique bien menée. Pendant vingt ans j'avais observer les hackers rêver, suer et construire, pour trop souvent constater que les pairs du vieux méchant IBM ou du nouveau méchant Microsoft repartaient nantis des récompenses concrètes. Pendant vingt ans j'avais vécu dans un ghetto un ghetto raisonnablement confortable, rempli de camarades intéressants, mais emmuré malgré tout derrière une vaste barrière de préjugés, intangible, annonçant : « ici, vous ne trouverez que des excentriques ».

L'annonce de Netscape avait lézardé cette barrière, pour au moins un court instant ; le monde des affaires avait été secoué dans sa complaisance sur l'idée qu'il se faisait des capacités des « hackers ». Mais les habitudes mentales paresseuses ont une inertie énorme. Si la société Netscape échouait, ou peut-être même si elle réussissait, l'expérience pourrait être perçue comme un fait unique et exceptionnel, qu'il serait inutile de tenter de reproduire. Et nous serions de nouveau parqués dans le même ghetto, aux murs un peu plus hauts qu'avant.

Pour éviter cela, il fallait que Netscape réussisse. Alors j'ai récapitulé ce que j'avais appris du mode de développement de type « bazar », et j'ai appelé la société Netscape, en leur proposant de les aider à développer leur licence et à mettre au point les détails de leur stratégie. Début février, j'ai pris l'avion pour Mountain View à leur demande, j'ai assisté à sept heures de réunions avec divers groupes au sein de leur quartier général, et je les ai aidés à développer les grandes lignes de ce qui deviendrait la licence publique de Mozilla et l'organisation correspondante.

J'ai profité de ma présence en ces lieux pour rencontrer plusieurs personnes

1. révolutionnaire américain

clés de la SiliconValley et de la communauté Linux des États-Unis d'Amérique (mais ces détails sont contés plus en détail sur la page d'histoire du site web de l'Open Source). Venir en aide à la société Netscape était clairement une priorité à court terme, et tous ceux à qui j'ai parlé avaient déjà compris la nécessité d'une stratégie à plus long terme, pour faire suite à la sortie de Netscape. Il était temps d'en mettre une au point.

15.5 Les origines du mouvement OpenSource

Les grandes lignes étaient faciles à deviner. Il nous fallait prendre les arguments pragmatiques et nouveaux que j'avais énoncés dans « CatB », les développer plus avant, et en faire une promotion poussée en public. Puisque les gens de Netscape avaient eux-mêmes intérêt à convaincre leurs investisseurs que cette stratégie n'était pas folle, on pouvait compter sur eux pour nous aider dans le cadre de cette promotion. Nous avons également très rapidement recruté Tim O'Reilly (et à travers lui, la société O'Reilly Associates).

Cependant, la véritable percée conceptuelle fut pour nous d'admettre qu'il nous fallait monter une campagne de mercatique et que cela mettrait en uvre des techniques de mercatique (conseil, construction d'une image, restructuration du concept) pour que tout cela fonctionne.

D'où le terme « OpenSource », que les premiers participants à ce qui deviendrait plus tard la campagne de l'OpenSource (et, finalement, l'organisation de l'« Initiative de l'OpenSource ») ont inventé lors d'une réunion tenue à Mountain View, dans les locaux de VA Research, le 3 février.

Il nous paraissait clair, en regardant en arrière, que le terme « free software » avait causé énormément de tort à notre mouvement au cours des années. Une partie en incombait à l'ambiguïté bien connue « free-speech/free-beer » [2]. Mais une partie plus importante provenait de quelque chose de pire l'association d'idées très répandue entre les termes « free software » et l'hostilité au droit de la propriété intellectuelle, le communisme, et d'autres idées que tout responsable de système d'information ne porte pas dans son cur.

Il était, et c'est toujours le cas, hors-sujet d'expliquer que la Free Software Foundation n'est pas hostile à toute propriété intellectuelle et que sa position n'est pas exactement celle d'une organisation communiste. Nous le savions. Mais nous avons réalisé, sous la pression de la sortie de Netscape, que la véritable position de la FSF ne comptait pas vraiment. Seul le fait que son évangélisme s'était retourné contre ses prédicateurs importait : désormais la presse professionnelle et l'industrie du logiciel associaient les mots « free software » aux stéréotypes négatifs exposés ci-dessus.

La réussite de notre initiative, suite au cas Netscape, ne serait possible que si on parvenait à remplacer les stéréotypes négatifs associés à la FSF par des stéréotypes positifs choisis par nous des contes pragmatiques, doux aux oreilles des gestionnaires et des investisseurs, parlant de fiabilité accrue, de coûts réduits, et de meilleures fonctionnalités.

En termes de mercatique conventionnelle, notre travail consistait à donner au produit une nouvelle image, et à lui construire une réputation qui donnerait

2. les anglais utilisent le même terme pour parler d'« expression libre » et de « bière gratuite », aussi le logiciel libre est-il souvent perçu comme du vulgaire « logiciel gratuit », ce qui fait fuir les industriels.

envie à l'industrie du logiciel de l'embrasser.

LinusTorvalds a accepté l'idée le lendemain de la réunion. On a commencé à travaillé sur le sujet quelques jours plus tard. Moins d'une semaine plus tard, BrucePerens avait enregistré le domaine opensource.org et avait mis en ligne la première version du site web de l'OpenSource. Il a aussi suggéré qu'on adopte en tant que "définition de l'OpenSource" les grandes lignes du logiciel libre mises au point par Debian, et il a commencé à enregistrer le terme « OpenSource » en tant que marque de certification de telle sorte qu'on puisse légalement exiger des gens qu'ils utilisent le terme « OpenSource » dans le cadre de produits conforme à cette définition.

Même les tactiques particulières, nécessaires pour mettre en place cette stratégie, m'ont paru claires dès ces premiers moments (et on les avait explicitement discutés au cours de la réunion initiale). Les points-clés :

Oublions la tactique de la conquête par le bas ; il faut convaincre la tête

> L'une des choses les plus claires était que la stratégie historique d'Unix, d'un évangélisme de conquête par le bas (reposant sur les ingénieurs qui convaincraient leurs patrons à l'aide d'arguments rationnels) avait été un échec. C'était une stratégie naïve, aisément démentie par la société Microsoft. De plus, la percée de la société Netscape ne provenait pas d'un ingénieur, mais d'un décideur stratégique (Jim Barksdale) qui avait compris tout cela et avait imposé sa vision des choses à ses subalternes.
>
> La conclusion s'imposait d'elle-même. Au lieu de conquérir la base, il nous faudrait cibler par notre discours les directions en visant directement les directions générales, techniques et informatiques.

Linux est notre meilleur cas d'école

> Il nous faut faire de Linux notre porte-étendard. Oui, on trouve d'autres exemples dans le monde de l'OpenSource, et la campagne leur rendra un hommage respectueux mais Linux a le nom le plus connu, la plus grande base installée, et la plus grande communauté de développeurs. Si Linux ne peut pas consolider la percée, rien d'autre, d'un point de vue pragmatique, n'a la moindre chance.

Suscitons l'intérêt des très grandes entreprises

> D'autres segments du marché dépensent plus d'argent (les PME/PMI et entreprises familiales en sont l'illustration la plus évidente) mais ces marchés sont plus diffus et difficiles à cibler. Les entreprises de Fortune500 ne se contentent pas de disposer de quantités d'argent phénoménales, elles les concentrent là où il est facile d'en approcher. C'est pourquoi l'industrie du logiciel est en grande partie aux ordres des Fortune500. C'est par conséquent le Fortune500 qu'il nous faut convaincre.

Mettons dans le coup les journaux prestigieux qui sont au service des Fortune500

> Le choix de cibler les Fortune500 implique de capter l'attention des médias qui influencent les plus importants décideurs et investisseurs ; pour être précis, il s'agit des publications suivantes : the New York

Times, the Wall Street Journal, the Economist, Forbes, et Barron's Magazine.

À ce sujet, il est nécessaire mais insuffisant de mettre dans le coup la presse technique informatique ; cela n'est utile qu'en tant que précondition pour insuffler la Bourse de WallStreet elle-même dans les media les plus en vue.

Inculquons aux hackers les tactiques de guérilla mercatique

Il était tout aussi clair que l'éducation de la communauté des hackers serait aussi importante que notre approche du monde réel. À quoi bon envoyer une poignée d'ambassadeurs tenir un discours efficace si, sur le terrain, la plupart des hackers en tenaient un autre, qui ne convaincrait personne sinon eux-mêmes ?

Utilisons la certification de l'OpenSource comme garantie de pureté

L'une des menaces qui nous attendait était la possibilité que le terme « Open Source » soit récupéré et amélioré par la Microsoft ou une autre société importante, le corrompant au passage, en annihilant notre message. C'est pour cette raison que BrucePerens et moi-même avons rapidement décidé d'enregistrer ce terme comme une marque de certification et de le lier à la définition de l'OpenSource (qui est une copie des grandes lignes du logiciel libre mises au point par Debian). Cela nous permettrait de dissuader les esprits chagrins sous la menace d'une poursuite en justice.

15.6 Révolutionnaire malgré moi

La mise au point de cette stratégie fut assez facile. Le plus dur (en tout cas, pour moi) fut d'accepter le rôle que j'aurais à y tenir.

J'avais compris une chose dès le début : c'est que la presse fait la sourde oreille aux abstractions. Ils n'écrivent rien sur des idées que ne défendent pas les personnalités les plus en vue du moment. Il faut que tout ne soit qu'histoires, drames, conflits, larmes et sang. Sans quoi, la plupart des journalistes retournent se coucher et s'ils continuent malgré tout, ce sont leurs rédacteurs en chef qui opposeront leur veto.

C'est pourquoi je savais qu'on aurait besoin d'une personne aux caractéristiques très précises pour faire front à la réaction de la communauté aux actions de Netscape. Il nous fallait un brandon, un porte-drapeau, un propagandiste, un ambassadeur, un évangéliste quelqu'un qui sache danser et chanter sur tous les toits, séduire les journalistes, fricoter avec les PDG, et asséner de grands coups sur la machine des médias jusqu'à ce que ses rouages tournent dans l'autre sens et proclament : « C'est la révolution ! »

À la différence de celui de la plupart des hackers, mon cerveau est celui d'un extraverti et j'ai déjà une solide expérience des contacts avec la presse. En examinant mon entourage, je n'ai trouvé personne de plus qualifié que moi pour tenir le rôle de l'évangéliste. Mais je ne voulais pas de ce travail, car je savais qu'il me mènerait la vie dure pendant des mois, peut-être des années. Je n'aurais plus d'intimité. La presse grand public me décrirait probablement comme un informaticien autiste et (ce qui est pire) je serais méprisé par une

proportion significative des miens, qui me considéreraient comme un vendu ou quelqu'un qui tire la couverture à lui. Et, pire que tout le reste, je n'aurais plus le temps de programmer !

Je devais me poser la question : en as-tu suffisamment marre d'observer ta tribu perdre les batailles pour faire ce qu'il en coûte de remporter la victoire ? J'ai décidé d'y répondre par l'affirmative et cela acquis, je me suis consacré à la tâche ingrate mais nécessaire de devenir une personnalité publique et un interlocuteur des médias.

J'avais appris quelques ficelles des médias alors que j'éditais *The New Hacker's Dictionary*. Cette fois-ci, j'ai pris le travail au sérieux, et j'ai développé toute une théorie de manipulation des médias que j'ai ensuite appliquée. Ce n'est pas l'endroit idéal pour l'exposer en détail, mais elle gravite autour de l'utilisation de ce que j'appelle une « dissonance séduisante » pour attiser une curiosité dévorante à l'encontre de l'évangéliste, et exploiter jusqu'au bout cette cette dernière pour faire passer les idées.

La combinaison de l'étiquette « OpenSource » et de ma promotion délibérée a eu les bonnes et mauvaises conséquences que j'escomptais. Dix mois après l'annonce de Netscape, on constate une croissance continue et exponentielle du nombre d'articles dans les médias traitant de Linux et du monde de l'Open-Source en général. Pendant toute cette période, environ un tiers de ces papiers me citaient directement ; la plupart des deux autres tiers faisaient appel à moi en tant que source indirecte d'informations. Au même moment, une minorité belliqueuse de hackers m'a traité d'intraitable égoïste. J'ai réussi à garder mon sens de l'humour et à plaisanter sur ces deux sujets (même si cela s'est parfois révélé difficile).

Depuis le début, mon plan consistait à confier le rôle de l'évangéliste à un successeur, un individuel ou une organisation. Le temps viendrait où le charisme personnel devrait céder la place à une respectabilité d'une institution plus répandue (et, en ce qui me concerne, le plus vite serait le mieux !). Au moment où je rédige ces lignes je tente de transférer mon carnet d'adresses personnel et la réputation que je me suis savamment construit auprès de la presse à l'Open-Source Initiative, une société à but non lucratif fondée dans le seul but de gérer la marque de certification OpenSource. J'en suis actuellement le président, mais j'espère ne pas le demeurer indéfiniment.

15.7 Les phases de la campagne

La campagne de l'OpenSource a débuté lors de la réunion de Mountain View et a rapidement mis en place un réseau informel d'alliés connectés par l'Internet (y compris des personnalités-clés de Netscape et d'O'Reilly Associates). Quand j'écris « nous », ci-dessous, je me réfère à ce réseau.

Du 3 février au jour où Netscape a effectivement publié son code source, le 31 mars, notre souci principal fut de convaincre la communauté des hackers que la marque « OpenSource » et les arguments qui lui étaient associés étaient la meilleure solution pour tenter de convaincre le grand public. Ils se sont révélés plus réceptifs que nous n'imaginions. En réalité, le désir refoulé d'un message moins dogmatique que celui de la Free Software Foundation était courant.

Quand la vingtaine de meneurs de la communauté présents au sommet du logiciel libre le 7 mars ont voté et adopté le terme « OpenSource », ils ont ratifié

formellement une tendance qui était déjà claire sur le terrain, parmi les développeurs. Six semaines plus tard, une majorité confortable de la communauté parlait notre langage.

En avril, suite au sommet et à la publication du code source de Netscape, notre souci principal fut de recruter autant de parents adoptifs que possible au mouvement de l'« OpenSource ». Le but était de rompre l'isolement de Netscape et de nous acheter une assurance au cas où Netscape fasse mauvaise figure et manque ses objectifs.

Ce fut la période la plus éprouvante. Les apparences étaient pourtant encourageantes : techniquement, Linux proposait les fonctionnalités les plus modernes les unes après les autres, le phénomène de l'OpenSource, plus général, bénéficiait d'une couverture croissante dans la presse informatique, et nous commencions à jouir d'une couverture positive dans la presse grand public. Cependant, j'avais douloureusement conscience du fait que notre réussite était encore fragile. Suite à une débauche initiale de contributions, la participation de la communauté au développement de Mozilla a beaucoup souffert de la nécessité de disposer de la bibliothèque Motif. Aucun des grands éditeurs indépendants de logiciels ne s'était encore engagé à porter son produit sur la plate-forme GNU/Linux. Netscape paraissait encore isolé, et son navigateur continuait à concéder des parts de marché à Internet Explorer. Un revers grave ne manquerait pas de faire les choux gras de la presse et de marquer l'opinion publique.

Notre première percée, suite à l'affaire Netscape, vint le 7mars quand la société Corel a annoncé qu'elle proposerait un ordinateur pour le réseau , Netwinder, fondé sur Linux. Mais cela ne suffisait pas ; pour nourrir la flamme, il nous fallait des engagements, non pas de la part de seconds couteaux désireux de gratter des parts de marché où ils pourraient les trouver, mais de la part de ceux qui mènent la danse dans leur propre branche. Ce sont donc les annonces des sociétés Oracle et Informix, à la mi-juillet, qui ont mis fin à cette période fragile.

Les pontes des bases de données avaient rejoint le parti de Linux trois mois plus tôt que je ne pensais, mais nous ne nous en sommes pas plaint. Nous nous étions demandés combien de temps pourrait durer l'aura positive de notre mouvement en l'absence d'engagements de la part d'éditeurs indépendants de logiciels (ÉIL, en anglais ISV), et notre nervosité allait croissant en attendant de telles déclarations. Après les annonces d'Oracle et d'Informix d'autres ÉIL ont annoncé les uns après les autres qu'ils proposeraient une version pour Linux de leurs produits, à tel point que c'en est devenu une routine et qu'on pourrait même survivre à un échec de l'expérience de Mozilla.

La phase de consolidation prit place de la mi-juillet à début novembre. C'est à cette époque que nous avons commencé à remarquer une couverture relativement régulière de la part des médias prestigieux que j'avais ciblés à l'origine, dont les têtes de gondole étaient des articles dans The Economist et un article annoncé sur la couverture de Forbes. Divers éditeurs de logiciels et fabriquants de matériels ont envoyé des gens prendre le pouls de la communauté de l'OpenSource et ont commencé à réfléchir à des stratégies pour profiter de ce nouveau modèle. Et de façon interne, le plus grand éditeur de logiciels fermés commençait à se poser sérieusement des questions.

À quel point, nous l'apprîmes avec précision quand les documents Halloween, désormais de sinistre réputation, filtrèrent hors de chez Microsoft.

Les documents Halloween étaient de la dynamite. C'était un témoignage éclatant à la gloire des forces à l'uvre dans le développement selon le modèle OpenSource, de la part de la société qui avait le plus à perdre de la réussite de Linux. Et ils ont confirmé nombre des soupçons les plus obscurs quant aux tactiques que Microsoft emploierait dans le but d'endiguer ce mouvement.

Les documents Halloween ont bénéficié d'une couverture massive dans la presse les premières semaines de novembre. Ils ont provoqué une nouvelle vague d'intérêt pour le phénomène de l'OpenSource, confirmant de façon fortuite et heureuse toutes les idées que nous tentions de faire passer depuis des mois. Et ils ont directement provoqué une invitation de votre serviteur à une conférence au cur d'un groupe trié sur le volet des investisseurs les plus importants de MerrillLynch, sur l'état de l'industrie du logiciel et sur les perspectives de l'OpenSource.

WallStreet, enfin, nous tendait les bras.

15.8 Les faits concrets

Alors que la campagne de l'OpenSource battait son plein dans les médias, en engageant une guerre virtuelle, les faits techniques et les phénomènes du marché, bien concrets, changeaient eux aussi. J'en passerai brièvement ici quelques-uns en revue car ils forment un tout intéressant avec les tendances de la presse et de la perception du phénomène par le grand public.

Durant les dix mois qui ont suivi la sortie de Netscape, Linux a continué d'accumuler les compétences techniques. Le développement d'une proposition solide pour le SMP et le nettoyage effectif du code 64 bits ont installé d'importantes fondations pour l'avenir.

La salle de linuxettes utilisée pour calculer les scènes d'images de synthèse du film « Titanic » a fait peur aux constructeurs de machines graphiques onéreuses. Puis le projet Beowulf, de construire des super-ordinateurs à partir de machines peu coûteuses, a démontré que la sociologie de Linux, sorte d'armée de fourmis fondée sur le modèle des petits ruisseaux, pouvait faire des grandes rivières, même dans le domaine hyper moderne du calcul scientifique.

Rien de significatif n'a projeté les concurrents OpenSource de Linux sous les feux de la rampe. Et les Unix propriétaires ont continué à perdre des parts de marché ; en fait, dès le début du second semestre, seuls les systèmes NT et Linux continuaient de rogner des parts de marché au sein des Fortune500, et Linux progressait plus rapidement.

Le logiciel Apache a confirmé son avance dans le marché des serveurs pour le web. En novembre, le navigateur de Netscape a renversé sa courbe de parts de marché et a commencé à reprendre du terrain à Internet Explorer.

15.9 Perspectives

J'ai relaté ici les événements récents en partie pour les consigner. De façon plus importante, cela met en place un décor qui peut nous servir à comprendre les tendances à court terme et faire quelques projections pour l'avenir (j'écris ces lignes à la mi-décembre 1998).

Voici tout d'abord quelques prédictions pour l'année prochaine :

- La population des développeurs selon le modèle OpenSource continuera d'exploser, et cette croissance sera alimentée par le prix sans cesse plus abordable des compatibles IBM PC et des connexions sur l'Internet.
- Linux continuera à mener la danse, la taille relative de sa communauté de développeurs compensant les compétences techniques en moyenne plus élevées des gens de l'OpenSource qui se consacrent aux projets BSD, et de la minuscule équipe travaillant sur le Hurd.
- Les ÉIL seront de plus en plus nombreux à proposer des solutions pour la plate-forme Linux ; les engagements des éditeurs de serveurs de bases de données marquèrent un tournant décisif. L'engagement de la société Corel de proposer une version complète de leur suite bureautique pour Linux montre la voie.
- La campagne de l'OpenSource volera de victoire en victoire et fera évoluer les consciences des directions générales, techniques, informatiques, et des investisseurs. Les directions subiront une pression sans cesse croissante d'utiliser des produits issus du monde de l'OpenSource, non pas de leurs subalternes, mais de leurs supérieurs.
- Les solutions discrètes de serveurs Samba sur plate-forme Linux remplaceront un nombre croissant de machines sous NT, même dans des entreprises dont la ligne de conduite est de n'utiliser que des produits Microsoft.
- La part de marché des Unix propriétaires continuera à s'affaiblir. L'un au moins des concurrents les plus faibles (probablement DG-UX ou HP-UX) en sera même réduit à déposer le bilan. Mais quand cela se produira, les analystes y verront plus l'uvre de Linux que celle de Microsoft.
- Microsoft ne proposera pas un système d'exploitation prêt pour l'entreprise, car MS-Windows2000 ne sera pas vendu sous une forme exploitable (avec 60 millions de lignes de code, ce nombre croissant encore, son développement n'est plus maîtrisable).

En extrapolant ces tendances on peut risquer quelques prédictions, plus audacieuses, à moyen terme (de 18 à 32mois d'ici) :

- Le fait de proposer des solutions techniques aux utilisateurs commerciaux de systèmes d'exploitation OpenSource sera un nouveau secteur d'activité, très rentable, provoqué par, et renforçant, leur utilisation dans le monde des affaires.
- Les systèmes d'exploitation OpenSource (Linux menant la marche) engloutiront le marché des fournisseurs d'accès à l'Internet (FAI), centres de données. NT ne pourra pas résister de manière efficace à ce changement ; le coût réduit, la disponibilité du code source, et une fiabilité sans défaillance, 24h/24 et 7j/7 formeront un tout irrésistible.
- Le secteur de l'Unix propriétaire s'écroulera presque entièrement. Il paraît raisonnable de supposer que le système Solaris survivra sur le matériel haut de gamme de Sun, mais la plupart des autres systèmes jouant dans la cour du logiciel propriétaire appartiendront bientôt au passé.
- MS-Windows2000 sera soit annulé soit mort-né. Quoi qu'il en soit, ce sera une catastrophe sans nom, le pire désastre stratégique qui soit jamais arrivé à la société Microsoft. Cela affectera malgré tout assez peu leur mainmise sur le marché des postes de travail, ces deux prochaines années.

Au premier coup d'il, ces tendances ressemblent à une recette pour ne laisser survivre que Linux. Mais la vie est compliquée (et Microsoft tire tant d'argent et de présence sur le marché suite à sa position dominante sur les postes de

travail qu'on ne peut pas le laisser pour compte, même après le déraillement de MS-Windows2000.

D'ici deux ans, ma boule de cristal devient plus laiteuse. L'avenir qui nous attend dépend de questions comme : le ministère de la justice des États-Unis d'Amérique scindera-t-il Microsoft ? Le système BeOS ou OS/2 ou MacOS/X, ou un autre créneau de système d'exploitation propriétaire, à moins qu'il ne s'agisse d'une conception radicalement novatrice, trouvera-t-il le chemin de l'Open-Source et se posera-t-il en concurrent efficace de la conception de Linux, qui a 30ans d'âge ? Les problèmes liés à l'an 2000 jetteront-ils l'économie mondiale dans une dépression telle que tout le monde pourra jeter ses prévisions ?

Ce sont là des impondérables. Mais une question demeure intéressante : la communauté Linux pourra-t-elle enfin fournir une bonne interface graphique à tout le système ?

Je pense que le scénario le plus probable pour une échéance à deux ans, est que Linux contrôlera les serveurs, les centres de données, les FAI, et l'Internet, alors que Microsoft gardera la mainmise du poste de travail. Ensuite, tout dépendra du fait que GNOME, KDE, ou une autre interface graphique (ainsi que les applications, construites ou reconstruites pour l'exploiter) devienne suffisamment bonne pour concurrencer Microsoft sur son propre terrain.

Si cela était avant tout un problème technique, l'issue ne ferait aucun doute. Mais ce n'est pas le cas ; c'est un problème de conception ergonomique et de psychologie de l'interface, et les hackers ont historiquement été mauvais à ce petit jeu-là. Ils sont très bons dans la conception d'interfaces réservées à d'autres hackers, mais assez mauvais dès qu'il s'agit de modéliser suffisamment bien les processus cognitifs des 95 restants de la population pour écrire des interfaces que M.Dupont et sa maman seraient prêts à acquérir.

Cette année fut l'année des applications ; il est désormais clair qu'on décidera suffisamment d'ÉIL pour obtenir celles que nous n'écrirons pas nous-mêmes. Je pense que ces deux prochaines années, le problème sera plutôt de croître suffisamment pour rattraper (et dépasser !) les normes de qualité en matière d'interfaces dictées par la société Macinstosh, et de combiner ces dernières avec les vertus d'un Unix traditionnel.

On plaisante à demi à propos de « devenir les maîtres du monde », mais la seule manière d'y parvenir est de rendre service à tout le monde. Il s'agit de plaire à M. Dupont et sa maman ; et cela impose de reconsidérer ce que nous faisons d'une manière radicalement différente, et de réduire sans pitié la complexité visible de l'environnement par défaut pour la restreindre à un strict minimum.

Les ordinateurs sont des outils au service des êtres humains. Il faut donc qu'à terme les défis inhérents à la conception de matériel et de logiciel reviennent à concevoir des objets pour les êtres humains — tous les êtres humains.

Ce chemin sera long, et difficile. Mais nous nous devons — et nous devons à autrui — de le suivre jusqu'au bout et de faire les choses correctement. Que l'Open Source soit avec toi !

Annexe A

Le Débat Tanenbaum - Torvalds

Vous trouverez dans cette annexe le texte du débat entre Tanenbaum et Linus connu dans la communauté sous le titre « Linux est dépassé ». Andrew Tanenbaum est un chercheur très respecté, qui a fort bien gagné sa vie en étudiant les systèmes d'exploitation et leur conception. Au début de 1992, ayant remarqué que les discussions sur Linux prenaient le pas sur les autres dans comp.os.minix, il décida qu'il était temps de publier ses opinions.

Bien qu'Andrew Tanenbaum fut tourné en ridicule pour sa maladresse et ses mauvais jugements sur le noyau Linux, la réaction qu'il essuya fut injuste. Lorsque Linus lui-même entendit dire que nous allions publier ce débat, il voulut s'assurer que tout le monde comprenne qu'il ne gardait aucune animosité envers Tanenbaum, et n'aurait en fait pas approuvé cette publication si nous n'avions pu le convaincre que cela montrerait comment les systèmes d'exploitation étaient perçus à l'époque.

La publication de cette annexe pourrait donner une bonne idée de l'état des choses dans les milieux universitaires au moment où Linus était sous pression pour avoir abandonné l'idée de micro-noyau. Le premier tiers de la discussion montre cela plus en détail.

Des copies électroniques de cette discussion sont faciles à trouver sur le Web. Il est amusant de lire ceci et de noter que parmi les personnes qui se joignirent à la discussion vous trouverez Ken Thompson (l'un des concepteurs d'Unix) et David Miller (devenu un hacker majeur du noyau Linux) et encore bien d'autres.

Pour replacer cette discussion dans son contexte, lorsqu'elle commença en 1992, le 386 était le microprocesseur dominant et le 486 n'était pas sorti sur le marché. Microsoft était encore une petite entreprise commercialisant MS-DOS et Word pour MS-DOS. Lotus 123 dominait le marché des tableurs, et WordPerfect celui des éditeurs de texte. DBASE était l'éditeur dominant de logiciels de gestion de base de données, et beaucoup d'entreprises dont les noms sont maintenant connus de tous (Netscape, Yahoo, Excite) n'existaient tout simplement pas.

De : ast@cs.vu.nl (Andy Tanenbaum)
Groupe de discussion : comp.os.minix
Sujet : Linux est dépassé
Date : 29 Jan 92 12 :12 :50 GMT

J'ai passé 15 jours aux États-Unis, je n'ai donc pas pu faire beaucoup de remarques sur Linux (non pas que j'en aurais dit plus, aurais-je été dans le coin), mais souhaite à présent publier un ou deux commentaires.

Comme la plupart d'entre vous le savent, MINIX est pour moi un passe-temps, quelque chose que je fais le soir quand j'en ai assez d'écrire des livres et qu'il n'y a pas de guerre exceptionnelle, de révolution, ou d'audience au sénat retransmise sur CNN. Mon vrai travail est professeur et chercheur dans le domaine des systèmes d'exploitation.

Étant donnée ma profession, je pense que je sais à peu près vers où l'exploitation va se diriger dans la décennie à venir. Deux aspects ressortent :

1. SYSTÈME MONOLITHIQUE CONTRE MICRO-NOYAU

La plupart des anciens systèmes d'exploitation sont monolithiques, c'est-à-dire que le système d'exploitation entier est un unique fichier a.out qui s'exécute en 'mode noyau'. Ce binaire contient la gestion des processus, la gestion de la mémoire, le système de fichiers et le reste. Des exemples de tels systèmes sont UNIX, MSDOS, VMS, MVS, OS/360, MULTICS, et beaucoup d'autres.

L'autre option est un système à micro-noyau, dans lequel le plus gros du SE s'exécute en tant que processus séparés, pour la plupart en dehors du noyau. Ils communiquent par passage de messages. Le travail du noyau est de contrôler le passage des messages, le contrôle des interruptions, la gestion des processus bas-niveau, et si possible les E/S. Des exemples de cette conception sont les RC4000, Amoeba, Chorus, Mach, et le pas-encore-réalisé MS-Windows/NT.

Bien que je pourrais me lancer ici dans une longue histoire sur les mérites relatifs des deux concepts, je me contenterai de dire que parmi les personnes concevant à l'heure actuelle des systèmes d'exploitation, la discussion est essentiellement close. Les micro-noyaux ont gagné. Le seul vrai argument pour les systèmes monolithiques était la performance, et il est maintenant assez évident que les systèmes à micro-noyau peuvent être tout aussi rapides que les systèmes monolithiques (par ex., Rick Rashid a publié des notes comparant Mach 3.0 à des systèmes monolithiques), et il ne reste donc maintenant que des détails à régler.

MINIX est un système à micro-noyau. Le système de fichiers et la gestion de la mémoire sont des processus séparés, s'exécutant en dehors du noyau. Les gestionnaires d'E/S sont aussi des processus séparés (dans le noyau, mais seule la nature comateuse des processeurs Intel fait qu'il est difficile de faire autrement). LINUX est un système de style monolithique. C'est un pas de géant en arrière dans les années 1970. C'est comme prendre un programme C existant et qui marche et le réécrire en BASIC. Pour moi, écrire un système monolithique en 1991 est une idée vraiment médiocre.

2. PORTABILITÉ

Il était une fois le microprocesseur 4004. En grandissant, il devint un 8008. Puis il subit une opération de chirurgie plastique et devint le 8080. Il engendra le 8086, qui engendra le 8088, qui engendra le 80286, qui engendra le 80386, qui engendra le 80486, et ainsi de suite jusqu'à la Nième génération. Pendant ce temps, les microprocesseurs RISC apparurent, et certains d'entre eux développant plus de 100 MIPS. Des vitesses supérieures à 200 MIPS seront vraisemblables atteintes durant les prochaines années. Ces choses ne vont pas disparaître subitement. Ils prennent peu à peu le pas sur la lignée des 80x86 et exécuteront de vieux programmes MS-DOS en simulant le 80386 au niveau logiciel (j'ai même écrit mon propre simulateur IBM PC en C, qui peut être obtenu par FTP depuis ftp.cs.vu.nl = 192.31.231.42 dans le répertoire minix/simulator). Je pense que c'est une erreur grossière que de concevoir un SE pour une architecture spécifique, alors que celle-ci ne sera bientôt plus disponible.

MINIX a été conçu pour être raisonnablement portable, et a été porté de la ligne Intel aux 680x0 (Atari, Amiga, Macintosh), SPARC, et NS32016. Linux est attaché de très près au 80x86. C'est une mauvaise approche.

Ne le prends pas mal, je ne suis pas mécontent de LINUX. Il va amener tous ceux qui veulent faire évoluer MINIX en BSD UNIX derrière moi. Mais en toute honnêteté, je voudrais suggérer à ceux qui veulent un SE « libre » MODERNE qu'ils recherchent un SE à micro-noyau, portable, comme peut-être GNU ou quelque chose comme ça.

Andy Tanenbaum (ast@cs.vu.nl)

P.S. Juste en passant, Amoeba a un émulateur UNIX (s'exécutant au niveau utilisateur), mais il est loin d'être complet. S'il y a quelqu'un qui veut travailler dessus, merci de me le faire savoir. Pour exécuter Amoeba, vous devez avoir quelques 386, l'un d'eux nécessitant 16M, et tous équipés d'une carte Ethernet WD.

De : torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Sujet : Re : LINUX est dépassé
Date : 29 Jan 92 23 :14 :26 GMT
Organisation : University of Helsinki

Bien, avec un sujet tel que celui-ci, j'ai bien peur de devoir répondre. Que les utilisateurs de minix qui en ont assez d'entendre parler de linux m'excusent. J'aimerais pouvoir dire « ne tenez pas compte de cette attaque », mais... il est temps d'entamer une sérieuse querelle !

Dans l'article <12595@star.cs.vu.nl> ast@cs.vu.nl (Andy Tanenbaum) écrit :

> J'ai passé 15 jours aux États-Unis, je n'ai donc pas pu faire beaucoup de remarques sur Linux (non pas que j'en aurais dit plus, aurais-je été dans le coin), mais souhaite à présent publier un ou deux commentaires.
>
> Comme la plupart d'entre vous le savent, MINIX est pour moi un passe-temps, quelque chose que je fais le soir quand j'en ai assez d'écrire des livres et qu'il n'y a pas de guerre exceptionnelle, de révolution, ou d'audience au sénat retransmise sur CNN. Mon vrai travail est professeur et chercheur dans le domaine des systèmes d'exploitation.

Vous utilisez cela [être un professeur] comme excuse pour les limitations de minix ? Désolé mais vous avez perdu : j'ai plus d'excuses que vous, et linux bat toujours minix à plate couture dans la plupart des domaines. Et je n'insisterai pas sur le fait que le plus gros de l'excellent code du minix pour PC semble avoir été écrit par Bruce Evans.

Re 1 : vous considérez minix comme un passe-temps - regardez qui gagne de l'argent avec minix, et qui offre linux. Maintenant parlons passe-temps. Faites en sorte que minix soit librement accessible, et l'un de mes plus gros griefs disparaîtra. Linux a été très certainement un passe-temps (mais du type meilleur type : sérieux) pour moi : je n'ai pas reçu d'argent, et il ne fait même pas partie de mes études universitaires. J'ai tout réalisé sur mon propre temps, et sur ma propre machine.

Re 2 : vous êtes professeur et chercheur : c'est une sacrée bonne excuse expliquant les dommages cérébraux provoqués par minix. J'espère seulement (et je le suppose) qu'Amoeba n'est pas aussi nul que minix.

1. SYSTÈME MONOLITHIQUE CONTRE MICRO-NOYAU

C'est vrai, linux est monolithique, et je suis d'accord que les micro-noyaux sont mieux. Sur un sujet appelant moins à polémique, j'aurais été probablement d'accord avec l'essentiel de ce que vous avez dit. D'un point de vue théorique (et esthétique), linux a perdu. Si le noyau GNU avait été fini au dernier printemps, je ne me serais même pas embêté à démarrer mon projet : le fait est qu'il ne l'était pas et ne l'est pas encore. Linux a l'énorme avantage d'être disponible maintenant.

> MINIX est un système à micro-noyau. [écourté, mais de façon que vous ne puissiez perdre le fil]. LINUX est un système monolithique.

Si c'était le seul critère de « bonne qualité » d'un noyau, ce serait exact. Ce que vous ne mentionnez pas, c'est que minix ne fonctionne pas comme un micro-noyau de manière correcte, et pose des problèmes avec le multitâche en temps réel (dans le noyau). Si j'avais fait un SE ayant des problèmes avec un système de fichiers subdivisé en unités d'exécution, je n'aurais pas été aussi prompt à condamner les autres : en fait, j'aurais fait tout mon possible pour que les autres oublient ce fiasco.

[Oui, je sais qu'il y a des bitouillages sur le multithreading pour minix, mais ce sont des bitouillages, et Bruce Evans m'a dit qu'il existe un tas de contraintes]

2. PORTABILITÉ

« La portabilité, c'est pour les gens qui ne savent pas écrire de nouveaux programmes » - moi, à l'instant (ton ironique).

Le fait est que linux est plus facilement portable que minix. Quoi ? vous entends-je dire. C'est vrai - mais pas dans le sens où l'entend ast : j'ai conçu linux aussi conforme qu'il m'était possible aux normes (sans avoir de copie des normes POSIX sous la main). Porter des choses sous linux est généralement *beaucoup* plus facile que de les porter sous minix.

Je suis d'accord avec le fait que la portabilité est une bonne chose : mais uniquement lorsque cela a réellement une signification. Cela n'a pas de sens de créer un système d'exploitation totalement portable : coller à une API portable suffit. La véritable *idée* d'un système d'exploitation est d'utiliser les fonctionnalités du matériel, et de les placer derrière une couche d'appels de haut niveau. C'est exactement ce que fait linux : il utilise en fait plus complètement les fonctionnalités du 386 que les autres noyaux. Bien sûr cela rend le noyau non portable au sens strict, mais cela conduit à une conception *vraiment* plus simple. Un compromis acceptable, mais qui a eu pour premier avantage de permettre l'existence de linux.

Je suis d'accord également avec le fait que linux porte la non-portabilité à un extrême : j'ai acquis mon 386 en janvier dernier, et linux faisait partie d'un projet visant à m'apprendre à m'en servir. Les choses auraient été faites de manière plus portable si j'avais eu un projet concret. Je ne cherche pas des tas d'excuses, bien que c'était une décision de conception, et quand j'ai démarré les choses en avril dernier, je ne pensais pas que quelqu'un l'utiliserait réellement. Je suis heureux de dire que j'ai eu tort car mes sources sont librement disponibles et chacun peut essayer de les porter, même si cela ne sera pas facile.

Linus

PS. Excusez-moi pour ce qui peut sembler parfois sévère : minix est suffisant si vous n'avez rien d'autre. Amoeba pourrait être impeccable si vous possédez 5 à 10 386 en secours, mais je ne m'y risquerai pas. Je ne me lance pas habituellement dans les querelles mais suis susceptible quand il s'agit de linux :)

De : ast@cs.vu.nl (Andy Tanenbaum)
Sujet : Re : LINUX est dépassé
Date : 30 Jan 92 13 :44 :34 GMT

Dans l'article <1992Jan29.231426.20469@klaava.Helsinki.FI>
torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds) écrit :

> Vous utilisez cela [être un professeur] comme excuse pour les limitations de minix ?

Les limitations de MINIX sont liées en partie au fait que je suis professeur : j'avais comme objectif précis de le faire fonctionner sur du matériel bon marché afin que les étudiants puissent se l'offrir. En particulier, durant des années il tourna sur un PC ordinaire à 4.77 MHz sans disque dur. Vous pouviez tout faire avec, modifier et recompiler le système inclus. Juste pour mémoire, jusqu'à il y a un an environ, il y avait deux versions, une pour le PC (disquettes 360K) et une pour le 286/386 (1.2M). La version PC se vendait deux fois mieux que la version 286/386. Je n'ai pas de chiffres, mais mon opinion est que la fraction des 60 millions de PC existants

qui sont des machines 386/486 opposées aux 8088/286/680x0 etc est faible. Parmi les étudiants elle est encore plus faible. Développer du logiciel gratuit uniquement destiné aux gens ayant assez d'argent pour acheter du matériel de première classe est un concept intéressant. Bien sûr, dans 5 ans, ça sera différent, mais dans 5 ans tout le monde emploiera le GNU gratuit sur sa station SPARC-5 à 200 MIPS, 64 Mo.

> Re 2 : vous êtes professeur et chercheur : c'est une sacrée bonne excuse expliquant les dommages cérébraux provoqués par minix. J'espère seulement (et je le suppose) qu'Amoeba n'est pas aussi nul que minix.

Amoeba n'a pas été conçu pour fonctionner sur un 8088 sans disque dur.

> Si c'était le seul critère de « bonne qualité » d'un noyau, ce serait exact. Ce que vous ne mentionnez pas, c'est que minix ne fonctionne pas comme un micro-noyau de manière correcte, et pose des problèmes avec le multitâche en temps réel (dans le noyau). Si j'avais fait un SE ayant des problèmes avec un système de fichiers subdivisé en unités d'exécution, je n'aurais pas été aussi prompt à condamner les autres : en fait, j'aurais fait tout mon possible pour que les autres oublient ce fiasco.

Un système de fichiers multi-thread est seulement un bitouillage d'amélioration des performances. Lorsqu'il y a seulement un processus actif, cas normal sur un petit PC, cela ne vous apporte rien et ajoute de la complexité au code. Sur des machines suffisamment rapides pour servir plusieurs utilisateurs, vous avez probablement assez de mémoire-cache pour assurer un bon cache à un bon taux, auquel cas le multi-thread ne vous apporte rien non plus. Cela ne rapporte vraiment que lorsque plusieurs processus font au même moment des E/S effectives sur disque. Que ça vaille la peine de rendre le système plus complexe dans ce cas est pour le moins discutable.

Je persiste à penser que concevoir un noyau monolithique en 1991 est une erreur fondamentale. Estime-toi heureux de ne pas être un de mes étudiants. Tu n'obtiendrais pas une bonne note pour une telle conception :-)

> Le fait est que linux est plus facilement portable que minix. Quoi ? vous entends-je dire. C'est vrai - mais pas dans le sens où l'entend ast : j'ai conçu linux aussi conforme qu'il m'était possible aux normes (sans avoir de copie des normes POSIX sous la main). porter des choses sous linux est généralement *beaucoup* plus facile que de les porter sous minix.

MINIX fut conçu avant POSIX, et est maintenant en train d'être (lentement) POSIXisé comme quiconque suivant ce groupe le sait. Tout le monde convient que des normes au niveau utilisateur sont une bonne idée. En aparté, je te félicite d'être capable d'écrire un système en conformité avec POSIX sans avoir les normes POSIX en face de toi. Je trouve ceci assez difficile après avoir étudié les normes dans le détail.

Je pense qu'écrire un nouveau système d'exploitation étroitement lié à un matériel particulier, surtout aussi étrange que la ligne Intel, est fondamentalement mauvais. Un SE lui-même doit être facilement portable vers de nouvelles plates-formes matérielles. Le fait qu'OS/360 fut écrit en langage d'assemblage IBM 360 il y a 25 ans peut probablement être excusé. Lorsque MS-DOS fut écrit explicitement pour le 8088 il y a 10 ans, c'était moins que brillant, comme le réalisent trop péniblement IBM et Microsoft maintenant seulement. Écrire un nouveau SE seulement

pour le 386 en 1991 te donne une autre mauvaise note pour ce trimestre. Mais si tu fais vraiment mieux à l'examen final, tu peux encore empocher le module.
Prof. Andrew S. Tanenbaum (ast@cs.vu.nl)

De : feustel@netcom.COM (David Feustel)
Sujet : Re : LINUX est dépassé
Date : 30 Jan 92 18 :57 :28 GMT
Organisation : DAFCO - An OS/2 Oasis

ast@cs.vu.nl (Andy Tanenbaum) écrit :

> Je persiste à penser que concevoir un noyau monolithique en 1991 est une erreur fondamentale. Estime-toi heureux de ne pas être un de mes étudiants. Tu n'obtiendrais pas une bonne note pour une telle conception :-)

C'est vrai. Einstein a eu des notes déplorables en maths et en physique.

De : pete@ohm.york.ac.uk (-Pete French.)
Sujet : Re : LINUX est dépassé
Date : 31 Jan 92 09 :49 :37 GMT
Organisation : Electronics Department, University of York, UK

dans l'article <1992Jan30.195850.7023@epas.toronto.edu>,
meggin@epas.utoronto.ca (David Megginson) dit :

> Dans l'article <1992Jan30.185728.26477feustel@netcom.COM> feustel@netcom.COM (David > Feustel) écrit :
>
> > C'est vrai. Einstein a eu des notes déplorables en maths et en physique.
>
> Et Dan Quayle une mauvaise note en sciences politiques. Je pense qu'il y a plus de Dan Quayle que d'Einstein ici bas.. ;-)

Quelle abominable réflexion ! Mais pour revenir au débat micro-noyau ↔ monolithique, n'utilise-t-on pas en partie un artifice de langage ? MINIX peut très bien être conçu comme un système micro-noyau, mais au bout du compte vous finirez toujours avec un gros morceau monolithique de données binaires constituant « le système d'exploitation ». N'est-il pas écrit en plusieurs programmes distincts parce que C n'accepte pas l'idée de plusieurs processus inclus dans un seul morceau de code monolithique. Y a-t-il vraiment une différence entre un micro-noyau écrit en plusieurs morceaux de C et un noyau monolithique écrit dans le style d'OCCAM ? Je suis enclin à penser que dans ce cas le concept monolithique serait meilleur que le concept micro-noyau, puisqu'avec l'avantage d'avoir un langage homogène, le noyau peut être conçu de manière encore plus modulaire que celui de MINIX.
MINOX cherche un défenseur :-)
-bat.

De : kt4@prism.gatech.EDU (Ken Thompson)
Sujet : Re : LINUX est dépassé
Date : 3 Fév 92 23 :07 :54 GMT

Organisation : Georgia Institute of Technology

Le point de vue peut être dissocié du côté utilitaire. Surtout si ce n'est pas la majorité du logiciel utilisé qui est peut-être dépassé, suivant le tout dernier critère de conception. La plupart des utilisateurs se soucieront peu du fait que la conception interne du système d'exploitation qu'ils utilisent est dépassée. Ils seront beaucoup plus intéressés par les performances et les possibilités au niveau utilisation.

Je serais plutôt d'accord avec le fait que les micro-noyaux vont probablement dans le sens de l'histoire. Cependant, à mon avis, il est plus facile d'implémenter un noyau monolithique. Et il est également plus facile pour lui de tourner rapidement en eau de boudin en cas de modification.

Salutations, Ken

De : kevin@taronga.taronga.com (Kevin Brown)
Sujet : Re : LINUX est dépassé
Date : 4 Fév 92 08 :08 :42 GMT
Organisation : University of Houston

Dans l'article <47607@hydra.gatech.EDU> kt4@prism.gatech.EDU (Ken Thompson) écrit :

> Le point de vue peut être dissocié du côté utilitaire. Surtout si ce n'est pas la majorité du logiciel utilisé qui est peut-être dépassé, suivant le tout dernier critère de conception. La plupart des utilisateurs se soucieront peu du fait que la conception interne du système d'exploitation qu'ils utilisent est dépassée. Ils seront beaucoup plus intéressés par les performances et les possibilités au niveau utilisation.
>
> Je serais plutôt d'accord avec le fait que les micro-noyaux vont probablement dans le sens de l'histoire. Cependant, à mon avis, il est plus facile d'implémenter un noyau monolithique. Et il est également plus facile pour lui de tourner rapidement en eau de boudin en cas de modification.

Quelle difficulté y a-t-il à structurer l'arborescence des programmes source d'un noyau monolithique en sorte que la plupart des modifications n'auront pas de répercussions vraiment négatives sur les sources ? Dans quelles sortes de pièges vous précipitez-vous avec ce type de comportement, et quelles suggestions pouvez-vous faire pour vous en sortir ?

Voilà ce que je demande : qu'y a-t-il de difficile à organiser les sources de manière à ce que la plupart des changements effectués dans le noyau restent localisés quant à leurs conséquences, même si le noyau est lui-même monolithique ?

Je pense que vous avez des années d'expérience sur les noyaux monolithiques :-), dès lors je pense que vous essayerez de répondre au mieux à de telles interrogations.

Kevin Brown

De : rburns@finess.Corp.Sun.COM (Randy Burns)
Sujet : Re : LINUX est dépassé
Date : 30 Jan 92 20 :33 :07 GMT
Organisation : Sun Microsystems, Mt. View, Ca.

Dans l'article <12615@star.cs.vu.nl> ast@cs.vu.nl (Andy Tanenbaum) écrit :

> Dans l'article <1992Jan29.231426.20469@klaava.Helsinki.FI> torvalds@klaava.Helsinki.>FI (Linus Benedict Torvalds) écrit :
> Bien sûr dans 5 ans ça sera différent, mais dans 5 ans tout le monde emploiera le GNU gratuit sur sa station SPARC-5 à 200 MIPS, 64M.

Et bien, je suis de ceux qui *aimeraient* que cela arrive.

> > Le fait est que linux est plus facilement portable que minix. Quoi ? vous entends-je dire. C'est vrai - mais pas dans le sens où l'entend ast : j'ai conçu linux aussi conforme qu'il m'était possible aux normes (sans avoir de copie des normes POSIX sous la main). Porter des choses sous linux est généralement *beaucoup* plus facile que de les porter sous minix.
> Je pense qu'écrire un nouveau système d'exploitation étroitement lié à un matériel particulier, surtout aussi étrange que la ligne Intel, est fondamentalement mauvais.

En premier lieu, les parties de Linux qui s'adaptent le mieux aux 80x86 sont le noyau et les pilotes de périphériques. Mon sentiment est que même si Linux n'est simplement qu'un pis-aller pour nous permettre d'employer des logiciels GNU, cela vaut encore la peine d'avoir un noyau bien adapté pour le plus grand nombre d'architectures existantes.

> Un SE lui-même doit être facilement portable vers de nouvelles plates-formes matérielles.

Bien, la seule partie de Linux qui ne soit pas portable est le noyau et les pilotes de périphériques. Par rapport aux compilateurs, aux utilitaires, aux systèmes de fenêtrage, etc. c'est vraiment une petite partie de l'effort. Étant donné que Linux possède une grande capacité de compatibilité dans les appels avec les SE portables, je ne m'en plaindrais pas. Je suis personnellement très heureux d'avoir un SE qui fait que certains d'entre nous pourront tirer avantage des logiciels provenant de Berkeley, FSF, CMU, etc.

Peut-être que dans 2 à 3 ans les variantes super économiques BSD et Hurd feront des petits, et Linux sera dépassé. Mais juqu'à présent Linux réduit de manière notable le coût d'utilisation des outils tels que gcc, bison, bash qui servent pour le développement d'un tel SE.

De : torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Sujet : Re : LINUX est dépassé
Date : 31 Jan 92 10 :33 :23 GMT
Organisation : University of Helsinki

> Dans l'article <12615@star.cs.vu.nl> ast@cs.vu.nl (Andy Tanenbaum) écrit :
> > Les limitations de MINIX sont liées en partie au fait que je suis professeur : j'avais comme objectif précis de le faire fonctionner sur du matériel bon marché afin que les étudiants puissent se l'offrir.

D'accord : c'est un point technique indiscutable, ce qui rend certains de mes commentaires inexcusables. Mais en même temps vous vous emmêlez un peu les pinceaux : maintenant vous admettez que certaines erreurs de minix proviennent de sa trop grande portabilité : y compris pour des machines qui n'étaient pas vraiment conçues pour unix. Cette supposition induit alors que minix ne pourrait pas

facilement être développé en y incluant des choses telles que la pagination, même avec des machines pouvant l'accepter. Oui, minix est portable, mais vous pouvez reformuler ceci comme « n'utilisant aucune fonctionnalité », et ce serait toujours vrai.

> Un système de fichiers multi-thread est seulement un bitouillage d'amélioration des performances.

Ce n'est pas vrai. C'est un bitouillage d'amélioration des performances *avec un micro-noyau*, mais cela devient automatique lorsque vous écrivez un noyau monolithique - c'est un domaine où les micro-noyaux ne sont pas très performants (comme je le faisais remarquer dans ma correspondance privée avec ast). Lorsque vous concevez un unix de manière « dépassée », vous obtenez automatiquement un noyau multithread : chaque processus fait son propre travail, et vous n'avez pas à faire de vilaines choses telles que des files d'attente de messages pour qu'il fonctionne de manière efficace.

De plus, il y a des gens qui considèrent « seulement un bitouillage d'amélioration des performances » comme vital : excepté le cas où vous possédez un cray-3, je pense que tout le monde se lasserait d'attendre le micro-ordinateur tout le temps. Je sais que je l'ai fait avec minix (bien sûr, je l'ai fait avec linux aussi, mais il est *beaucoup* mieux).

> Je persiste à penser que concevoir un noyau monolithique en 1991 est une erreur fondamentale. Estime-toi heureux de ne pas être un de mes étudiants. Tu n'obtiendrais pas une bonne note pour une telle conception :-)

Bien, je n'aurais pas eu de très bonnes notes même avec vous : j'ai eu une discussion (sans rapport aucun - même pas avec les SE) avec la personne qui enseigne ici à l'université la conception des SE. Je me demande quand j'apprendrai :)

> Je pense qu'écrire un nouveau système d'exploitation étroitement lié à un matériel particulier, surtout aussi étrange que la ligne Intel, est fondamentalement mauvais. Un SE lui-même doit

Mais *mon* point de vue est que le système d'exploitation *n'est pas* lié à un type quelconque de microprocesseur : UNIX fonctionne sur la plupart des processeurs existants. Oui, *l'implémentation* dépend de l'architecture, mais c'est une ÉNORME différence. Vous citez OS/360 et MS-DOG comme exemples de mauvaise conception car ils sont dépendants du matériel, et je suis d'accord avec vous. Mais il y a une grande différence entre ceux-ci et linux : l'API de linux est portable (non pas parce que je l'ai conçu astucieusement, mais parce que j'ai décidé de faire un unix plutôt-bien-pensé et testé).

Si vous écrivez des programmes pour linux à ce jour, vous ne devriez pas avoir trop de surprises en les recompilant pour Hurd au 21ème siècle. Comme remarqué (et pas seulement par moi), le noyau linux est une toute petite partie d'un système complet : les sources complètes de linux prennent environ 200 Ko sous forme compactée - les sources complètes d'un système en développement prennent au moins 10 Mo sous forme compactée (et facilement beaucoup, beaucoup plus). Et les sources complètes sont portables, sauf pour le minuscule noyau que vous pouvez (pour preuve : je l'ai fait) réécrire totalement à partir de rien en moins d'un an sans avoir *aucune* connaissance préalable. De fait, le noyau linux *complet* est beaucoup plus petit que ce qui concerne le 386 dans mach : i386.tar.Z pour la version actuelle

de Mach fait plus de 800 Ko compactés (83391 octets selon nic.funet.fi). Il est vrai que mach est « quelque peu » plus gros et a plus de fonctionnalités, mais cela devrait toujours vous dire quelque chose.

Linus

De : kaufman@eecs.nwu.edu (Michael L. Kaufman)
Sujet : Re : LINUX est dépassé
Date : 3 Fév 92 22 :27 :48 GMT
Organisation : EECS Department, Northwestern University

J'ai essayé d'envoyer ces deux réponses depuis mon travail, mais pense qu'elles ont disparu. Désolé si vous les avez déjà lues.

Andy Tanenbaum écrit un article intéressant (de même, il est intéressant de savoir qu'il lit effectivement ce groupe de discussion) mais je pense qu'il oublie un point important.

Il a écrit :

> Comme la plupart de vous le savent, pour moi MINIX est un passe-temps

Ce qui est probablement vrai pour la plupart des personnes, si ce n'est toutes, qui travaillent sur Linux. Nous ne développons pas un système pour conquérir le marché des systèmes d'exploitation, nous passons seulement un bon moment.

> Ils prennent peu à peu le pas sur la lignée des 80x86 et exécuteront de vieux programmes MS-DOS en simulant le 80386 au niveau logiciel

Bien, quand ceci arrivera et si je désire toujours jouer avec Linux, je pourrai le lancer sur mon simulateur de processeur 386.

> MINIX a été conçu pour être raisonnablement portable, et a été porté de la ligne Intel aux 680x0 (Atari, Amiga, Macintosh), SPARC, et NS32016. Linux est attaché de très près au 80x86. C'est une mauvaise approche.

C'est très bien pour les personnes qui possèdent ces machines, mais cela ne s'est pas passé sans dommage. Cette portabilité fut atteinte au détriment des performances et de quelques caractéristiques propres au 386. Avant de décréter que Linux n'est pas sur le bon chemin, vous devriez penser à quoi il va servir. Je vais l'utiliser sur mon 486 pour lancer des applications graphiques très gourmandes en calcul et en mémoire.

> Mais en toute honnêteté, je voudrais suggérer à ceux qui veulent un SE « libre » MODERNE qu'ils recherchent un SE à micro-noyau, portable, comme peut-être GNU ou quelque chose comme ça.

Je ne connais pas de système d'exploitation micro-noyau portable et libre. GNU Hurd est toujours dans les limbes et va le rester pour un moment encore. En avez-vous un à recommander, ou bien me taquinez-vous ? ;-)

Dans l'article <12615@star.cs.vu.nl> ast@cs.vu.nl (Andy Tanenbaum) écrit :

> Je pense qu'écrire un nouveau système d'exploitation étroitement lié à un matériel particulier, surtout aussi étrange que la ligne Intel, est fondamentalement mauvais. Un SE lui-même doit être facilement portable vers de nouvelles plates-formes matérielles.

Je pense comprendre nos points de désaccord. Vous regardez la conception de systèmes d'exploitation comme une fin en soi. Minix est bien car il est portable, micro-noyau, etc. Linux est mauvais car il est hermétiquement attaché à Intel. Ce n'est pas une étrange attitude venant de quelqu'un du monde académique, mais vous ne devriez pas vous attendre à la voir universellement partagée. Linux n'est pas conçu comme un outil pédagogique, ou un exercice abstrait. Il est écrit pour permettre aux gens d'exécuter des logiciels GNU aujourd'hui. Le fait qu'il ne puisse pas être utilisé dans cinq ans est moins important que le fait qu'aujourd'hui, je peux lancer dessus toutes sortes de logiciels que je veux. Vous maintenez que Minix est meilleur, mais s'il ne permet pas d'exécuter les logiciels que je veux, il n'est pas du tout bon (pour moi).

> Le fait qu'OS/360 fut écrit en langage d'assemblage IBM 360 il y a 25 ans peut probablement être excusé. Lorsque MS-DOS fut écrit explicitement pour le 8088 il y a 10 ans, c'était moins que brillant, comme le réalisent trop péniblement IBM et Microsoft maintenant seulement.

Même point. Microsoft n'est pas sorti avec DOS pour « explorer les frontières des systèmes d'exploitation ». Ils l'ont écrit pour faire de l'argent. En considérant le fait que MS-DOS se vend en plus grande quantité que tout le reste réuni, vous ne pouvez pas dire qu'ils ont raté leur but. Ce n'est pas que MS-DOS soit le meilleur système d'exploitation suivant d'autres caractéristiques, mais il a servi leur besoin.

Michael

De : julien@incal.inria.fr (Julien Maisonneuve)
Sujet : Re : LINUX est dépassé
Date : 3 Fév 92 17 :10 :14 GMT

J'aimerais apporter de l'eau au moulin de Kevin Brown pour la plupart de ses remarques. J'ajouterai quelques considérations propres à l'utilisateur :

- Quand ast affirme que le multi-thread pour un système de fichiers est inutile, cela me rappelle toutes les fois où j'ai voulu laisser une tâche en arrière-plan (comme la lecture d'une archive depuis une disquette), c'est tout simplement inutilisable, l'opérateur du shell aurait pu tout aussi bien être laissé de côté.
- Les utilitaires les plus intéressants ne sont même pas compilables sous Minix en raison des limitations incroyables du compilateur d'ATK. On pouvait les admettre à la rigueur sur un PC de base, sur un 386, cela devient absurde. N'importe quel bête compilateur DOS est doté d'un modèle mémoire 'large' (plus cher, d'accord). Je déteste le compactage 13 bits !
- L'absence de gestion de la mémoire virtuelle interdit d'étudier ce domaine à des fins d'expérimentation et d'utiliser des programmes de taille importante. La conception étrange du MM le rend également difficile à modifier.

Le problème est que même des travaux d'exploration sont pénibles sous minix. Si vous voulez accomplir une tâche (ou même vous amuser), même DOS devient un meilleur choix (avec des choses telles que DJ GPP). À la base, ce n'est rien d'autre qu'un exemple didactique de SE, un beau jouet, certes, mais un jouet quand même. Récupérer et appliquer des patches est fastidieux et exclut les mises à jour ultérieures.

Dommage quand il manque si peu pour en faire quelque chose de vraiment bien. Merci pour le travail Andy, mais Linux ne méritait pas votre réponse. Pour le commun des mortels, il fait beaucoup de choses mieux que Minix.

Julien Maisonneuve.

Ce n'est pas une descente en flamme, seulement mon expérience.

De : richard@aiai.ed.ac.uk (Richard Tobin)
Sujet : Re : LINUX est dépassé
Date : 4 Fév 92 14 :46 :49 GMT
Répondre à : richard@aiai.UUCP (Richard Tobin)
Organisation : AIAI, University of Edinburgh, Scotland

Dans l'article <12615@star.cs.vu.nl> ast@cs.vu.nl (Andy Tanenbaum) écrit :

> Un système de fichiers multi-thread est seulement un bitouillage d'amélioration des performances. Lorsqu'il y a seulement un processus actif, cas normal sur un petit PC, cela ne vous apporte rien

Je trouve l'utilisation d'un système de fichiers à thread unique particulièrement pénible avec Minix. J'ai souvent besoin de faire autre chose pendant que je lis des fichiers depuis une (atrocement lente) disquette. Je préfère jouer au solitaire pendant les grosses compilations de C ou de Lisp. J'aime bien examiner des fichiers dans le tampon d'un éditeur tout en compilant dans un autre.

(Le problème serait sans doute moindre si le système de fichiers se contentait de servir des fichiers au lieu d'interagir avec les entrées/sorties de terminal).

Évidemment, sur du Minix de base, sans consoles virtuelles ni aucune possibilité de lancer emacs, ce n'est pas vraiment un problème. Mais pour la plupart des gens, c'est un inconvénient, pas un avantage. Ce n'est pas simplement le fait que sur des machines mono-utilisateur il n'y a pas besoin de plus d'un processus actif. Cette idée n'est envisageable que parce que tant de gens sont habitués à des machines médiocres dotées de systèmes d'exploitation médiocres.

Quant à la portabilité, Minix ne l'emporte que grâce à ses prétentions limitées. Si vous vouliez un Unix complet avec pagination, ordonnanceur, système de fenêtrage et ainsi de suite, serait-il plus rapide de partir d'un Minix de base et d'ajouter ces fonctionnalités, ou depuis Linux et de s'occuper des spécificités du 386 ? Je ne crois pas qu'il soit juste de critiquer Linux quand ses objectifs sont si différents de ceux de Minix. Si on veut un système à but pédagogique, Minix est la réponse. Mais s'il s'agit d'avoir un environnement aussi proche que possible de celui d'une Sun (par exemple) sur son ordinateur personnel, il a quelques lacunes.

– Richard

De : ast@cs.vu.nl (Andy Tanenbaum)
Sujet : Re : LINUX est dépassé
Date : 5 Fév 92 14 :48 :48 GMT
Organisation : Fac. Wiskunde & Informatica, Vrije Universiteit, Amsterdam

Dans l'article <6121@skye.ed.ac.uk> richard@aiai.UUCP (Richard Tobin) écrit :

> Si vous vouliez un Unix complet avec pagination, ordonnanceur, système de fenêtrage et ainsi de suite, serait-il plus rapide de partir d'un Minix de base et d'ajouter ces fonctionnalités, ou depuis Linux et de s'occuper des spécificités du 386 ?

Une autre option qui semble être totalement oubliée ici est d'acheter UNIX ou un clone. Si vous voulez juste UTILISER le système, plutôt que de bitouiller dans ses entrailles, vous n'avez pas besoin du code source. Coherent est à seulement 99$, et

il y a plusieurs vrais systèmes UNIX avec plus de fonctionnalités pour plus d'argent. Pour le vrai bitouilleur, ne pas avoir le code source est fatal, mais pour ceux qui veulent juste un système UNIX, il y a beaucoup d'autres options (non gratuites).

Andy Tanenbaum (ast@cs.vul.nl)

De : ajt@doc.ic.ac.uk (Tony Travis)
Sujet : Re : LINUX est dépassé
Date : 6 Fév 92 02 :17 :13 GMT
Organisation : Department of Computing, Imperial College, University of London, UK.

ast@cs.vu.nl (Andy Tanenbaum) écrit :

> Une autre option qui semble être totalement oubliée ici est d'acheter UNIX ou un clone. Si vous voulez juste UTILISER le système, plutôt que de bitouiller dans ses entrailles, vous n'avez pas besoin du code source. Coherent est à seulement 99$, et il y a plusieurs vrais systèmes UNIX avec plus de fonctionnalités pour plus d'argent. Pour le vrai bitouilleur, ne pas avoir le code source est fatal, mais pour ceux qui veulent juste un système UNIX, il y a beaucoup d'autres options (non gratuites).

Andy, j'ai suivi le développement de Minix depuis que les premiers articles ont été postés dans ce groupe et emploie actuellement la version 1.5.10 avec les correctifs de Bruce Evans pour le 386.

Je veux 'simplement' un Unix sur mon PC et bitouiller ses entrailles ne m'intéresse pas, mais je **veux** le code source !

La philosophie qui consiste à créer à partir du travail des autres est un des grands principes sous-jacents qui explique le succès et la popularité d'Unix.

Cette philosophie repose sur la disponibilité du code source afin qu'il puisse être examiné, modifié et réutilisé pour des nouveaux programmes.

Il y a des années de cela, j'étais l'heureux bénéficiaire d'une licence pour le code source Unix ATT 7ème édition mais, même alors, j'ai accueilli votre décision de rendre celui de Minix disponible comme une libération par rapport aux entraves du copyright ATT ! !

Il me semble que vous devez parfois oublier que votre 'hobby' a eu un effet déterminant sur la disponibilité d'un Unix 'personnel' (c-à-d financièrement accessible) et que le PC 8086 sur lequel tournait Minix 1.2 m'a coûté en fait bien plus que le clone 386/SX que j'utilise à présent.

Il est clair que Minix *n'est pas* la panacée, mais je vois l'évolution vers des versions 386 et le 68000 ou d'autres architectures à espace d'adressage linéaire d'une manière assez semblable : c'est une bonne chose pour les gens comme moi qui utilisent Minix et se sentent gênés par l'architecture segmentée de la version PC dans le domaine des applications.

RIEN de ce que vous pouvez dire ne pourrait me convaincre d'utiliser Coherent...

Tony

De : richard@aiai.ed.ac.uk (Richard Tobin)
Sujet : Re : LINUX est dépassé
Date : 7 Fév 92 14 :58 :22 GMT
Organisation : AIAI, University of Edinburgh, Scotland

Dans l'article <12696@star.cs.vu.nl> ast@cs.vu.nl (Andy Tanenbaum) écrit :

> Si vous voulez juste UTILISER le système, plutôt que de bitouiller dans ses entrailles, vous n'avez pas besoin du code source.

Beaucoup d'entre nous souhaitent pouvoir bitouiller dans les entrailles du système. . .Vous serez débarrassé de la plupart d'entre nous quand BSD-detox ou GNU sortiront, ce qui devrait arriver dans les tous prochains mois (ouais, exactement).
– Richard

De : comm121@unixg.ubc.ca (Louie)
Sujet : Re : LINUX est dépassé
Date : 30 Jan 92 02 :55 :22 GMT
Organisation : University of British Columbia, Vancouver, B.C., Canada

Dans <12595@star.cs.vu.nl> ast@cs.vu.nl (Andy Tanenbaum) écrit :

> Mais en toute honnêteté, je voudrais suggérer à ceux qui veulent un SE « libre » MODERNE qu'ils recherchent un SE à micro-noyau, portable, comme peut-être GNU ou quelque chose comme ça.

Il n'y a réellement pas d'autre choix que Linux pour les gens qui, comme moi, veulent un SE « libre ». Considérant que la majorité des personnes susceptibles d'opter pour un SE « libre » utilisent le 386, la portabilité n'est vraiment pas un si gros problème. Si j'avais une Sparc, j'utiliserais Solaris.

Jusqu'ici, j'ai installé Linux avec gcc, emacs 18.57, kermit et tous les utilitaires GNU sans le moindre ennui. Nul besoin d'appliquer des correctifs. J'ai simplement suivi les instructions d'installation. Je ne peux pas trouver un tel SE **n'importe où** au même prix pour mes devoirs d'informatique. Et il semble que la gestion du réseau ainsi que X-Window seront portés sous Linux bien avant Minix. C'est quelque chose qui serait vraiment utile. À mon avis, la portabilité des programmes Unix standards est également importante.

Je sais que le concept du système monolithique n'est pas aussi bon que celui du micro-noyau. Mais à court terme (et je sais que je ne voudrais/pourrais pas faire évoluer mon 386), Linux me convient parfaitement.

Philip Wu pwu@unixg.ubc.ca

De : dgraham@bmers30.bnr.ca (Douglas Graham)
Sujet : Re : LINUX est dépassé
Date : 1 Fév 92 00 :26 :30 GMT
Organisation : Bell-Northern Research, Ottawa, Canada

Dans l'article <12595@star.cs.vu.nl> ast@cs.vu.nl (Andy Tanenbaum) écrit :

> Bien que je pourrais me lancer ici dans une longue histoire sur les mérites relatifs des deux concepts, je me contenterai de dire que parmi les personnes concevant à l'heure actuelle des systèmes d'exploitation, la discussion est essentiellement close. Les micro-noyaux ont gagné.

Pouvez-vous recommander une documentation (sans parti pris) qui traite des forces et faiblesses des deux approches ? Je suis certain qu'il y a quelque chose à dire sur l'approche micro-noyau, mais je m'interroge sur la similitude de Minix vis à vis des autres systèmes qui utilisent cette approche. Bien sûr, Minix fait appel

à beaucoup de tâches et de messages, mais il en faudrait plus pour qu'il ait une architecture de micro-noyau. Je suspecte que le code Minix ne soit pas découpé en tâches de manière optimale.

> micro-noyaux ont gagné. Le seul vrai argument pour les systèmes monolithiques était la performance, et il est maintenant assez évident que les systèmes à micro-noyau peuvent être tout aussi rapides que les systèmes monolithiques (par ex., Rick Rashid a publié des notes comparant Mach 3.0 à des systèmes monolithiques), et il ne reste donc maintenant que des détails à régler.

Mon principal grief envers Minix ne concerne pas ses performances. Ajouter des fonctionnalités est un calvaire - ce que, je suppose, une architecture micro-noyau est supposée alléger.

> MINIX est construit autour d'un micro-noyau.

Y a-t-il un consensus à ce sujet ?

> LINUX est un système de style monolithique. C'est un pas de géant en arrière dans les années 1970. C'est comme prendre un programme C existant et qui marche et le réécrire en BASIC. Pour moi, écrire un système monolithique en 1991 est une idée vraiment médiocre.

C'est une affirmation pertinente, mais je cherche encore à voir une raison d'être pour cela. Je crois que Linux consiste seulement en 12000 lignes de codes. Je ne vois pas en quoi le diviser en tâches et en messages à droite et à gauche l'améliorerait.

> Ne le prends pas mal, je ne suis pas mécontent de LINUX. Il va amener tous ceux qui veulent faire évoluer MINIX en BSD UNIX derrière moi. Mais en toute honnêteté, je voudrais suggérer à ceux qui veulent un SE « libre » MODERNE qu'ils recherchent un SE à micro-noyau, portable, comme peut-être GNU ou quelque chose comme ça.

Bien, il n'y a pas d'autres choix sur lesquels je puisse me porter en ce moment. Mais lorsque le SE de GNU viendra, j'aimerais bien reprendre ce bateau. Je sens que vous **êtes** quelque peu insatisfait de Linux (et cela me surprend un peu). Je suis enclin à penser que la raison pour laquelle tant de personnes s'y intéressent est qu'il offre plus de possibilités. Votre approche sur les gens désirant des fonctionnalités dans Minix a été de dire qu'ils n'en voulaient pas vraiment. Je prétends que l'exode vers Linux prouve que vous avez tort.

Avertissement : je n'ai rien à voir avec le développement de Linux. Je pense simplement que c'est un système plus facile à comprendre que Minix. – Doug Graham dgraham@bnr.ca Mes opinions sont les miennes propres.

De : hedrick@klinzhai.rutgers.edu (Charles Hedrick)
Sujet : Re : LINUX est dépassé
Date : 1 Fév 92 00 :27 :04 GMT
Organisation : Rutgers Univ., New Brunswick, N.J.

L'histoire du logiciel montre qu'à chaque fois la disponibilité l'emporte sur la qualité technique. Voilà le principal

avantage de Linux. C'est un petit système pour 386, tout-à-fait compatible avec l'Unix d'origine, et qui est disponible librement. J'ai laissé tombé la communauté

Minix il y a quelques années quand il devint évident que (1) Minix ne tirerait profit de rien à part du 8086 dans le futur proche, et (2) que la licence - bien qu'étonnamment bienveillante - le rendait toujours difficile d'accès pour ceux qui voudraient produire une version pour le 386. Apparemment plusieurs personnes ont effectué un bon travail pour le 386. Mais elles ne pouvaient distribuer que des fichiers diffs. Cela rend le travail sur un 386 peu pratique pour un nouvel utilisateur, et de fait je n'étais pas sûr de vouloir le faire.

Excusez-moi si les choses ont changé ces dernières années. S'il est maintenant possible d'avoir une version 386 prête à l'emploi, la communauté a développé un moyen de distribuer les sources Minix, et que des programmes Unix normaux est dorénavant devenu plus facile, alors je reconsidérerai ma position en ce qui concerne Minix. J'aime bien sa conception.

Il est possible que Linux soit rattrapé par Gnu ou un BSD libre. Cependant, si le SE Gnu suit l'exemple de tous les autres logiciels Gnu, il aura besoin de 128 Mo de mémoire et d'un disque de 1 Go. Il y aura toujours de la place pour un petit système. Mon système idéal serait le 4.4 BSD, mais la date de la version du 4.4 a considérablement dérapé dans le temps. Et puis avec toute l'équipe migrant vers BSDI, il est difficile de croire que la situation va aller en s'améliorant. Pour mon usage personnel, le système BSDI sera certainement très bien. Mais même leur prix très attractif est vraiment trop élevé pour la plupart de nos étudiants, et bien que les utilisateurs puissent en acquérir les sources, le fait que certaines soient propriétaires signifie encore que vous ne pourrez que mettre du code modifié sur les sites FTP accessibles au public. De toutes façons Linux existe et tout le reste n'est que littérature.

De : tytso@athena.mit.edu (Theodore Y. Ts'o)
Sujet : Re : LINUX est dépassé
Date : 31 Jan 92 21 :40 :23 GMT
Organisation : Massachusetts Institute of Technology

> En réponse à : message d'ast@cs.vu.nl du 29 Jan 92 12 : 12 :50 GMT
>
> > De : ast@cs.vu.nl (Andy Tanenbaum)
> > dans le répertoire minix/simulator). Je pense que c'est une erreur grossière que de concevoir un SE pour une architecture spécifique, alors que celle-ci ne sera bientôt plus disponible.

Ce n'est pas de votre faute si vous croyez que Linux est limité à l'architecture 80386, puisque de nombreux adeptes de Linux (y compris Linus lui-même) sont partis de ce principe. Néanmoins, la quantité de code spécifique 80386 n'est probablement pas plus importante que dans une implémentation Minix, et il n'y a certainement pas plus de code spécifique 80386 dans Linux que de code spécifique Vax dans BSD 4.3.

Ceci dit, le portage sur d'autres architectures n'a pas encore été réalisé. Mais si je commençais à porter un système de type Unix vers une nouvelle architecture, je commencerais probablement par Linux plutôt que Minix, simplement parce que je veux avoir un certain contrôle sur ce que je pourrais faire avec le système résultant une fois que j'aurais fini. Oui, j'aurais à réécrire de grandes portions du VM et des couches des gestionnaires de périphériques – mais j'aurais à le faire avec n'importe quel SE. Cela serait peut-être un petit peu plus difficile que pour porter Minix sur

la nouvelle architecture, mais cela ne serait probablement vrai que pour la première architecture cible du portage de Linux.

> Bien que je pourrais me lancer ici dans une longue histoire sur les mérites relatifs des deux concepts, je me contenterai de dire que parmi les personnes concevant à l'heure actuelle des systèmes d'exploitation, la discussion est essentiellement close. Les micro-noyaux ont gagné. Le seul vrai argument pour les systèmes monolithiques était la performance, et il est maintenant assez évident que les systèmes à micro-noyau peuvent être tout aussi rapides que les systèmes monolithiques (par ex., Rick Rashid a publié des notes comparant Mach 3.0 à des systèmes monolithiques), et il ne reste donc maintenant que des détails à régler.

Ce n'est pas forcément le cas. Je pense que vous peignez une vue plus manichéenne de l'univers qu'il ne l'est. Je vous renvoie à des articles du genre de celui de Brent Welsh « le système de fichiers est intrinsèque au noyau » dans lequel il estime que le système de fichiers est une abstraction suffisamment mature pour qu'elle réside dans le noyau, et non en-dehors comme ce serait le cas dans une conception micro-noyau stricte.

Un certain nombre de gens sont préoccupés par la rapidité de OSF/1 Mach comparée à celle des systèmes monolithiques ; plus spécialement, le nombre de changements de contexte nécessaires à la gestion du trafic réseau, et des systèmes de fichiers réseau en particulier.

Je connais les bénéfices d'une approche micro-noyau. Néanmoins, le fait reste que Linux est là, et que GNU non - et que les gens ont travaillé sur Hurd beaucoup plus longtemps que Linus n'a travaillé sur Linux. Minix ne compte pas parce qu'il n'est pas libre. :-)

Je suspecte que le choix entre micro-noyaux et noyaux monolithiques dépend de ce que l'on en fait. Si l'on s'intéresse à la recherche, il est évidemment bien plus facile de retirer et de remplacer des modules dans un micro-noyau, et puisque seuls les chercheurs écrivent des articles sur les systèmes d'exploitation, ipso facto les micro-noyaux doivent être la bonne approche. Néanmoins, je connais des tas de gens qui ne sont pas chercheurs, mais au contraire des programmeurs noyau pratiques, dont les préoccupations principales sont dans le coût de copie et de changement de contexte inhérents à un micro-noyau.

Pour autant, je n'adhère pas à votre argument selon lequel un système de fichiers multi-thread n'est pas nécessaire sur un système mono-utilisateur. Une fois lancé un système de fenêtrage, ainsi qu'une compilation dans une fenêtre, un lecteur de courrier dans une autre et les nouvelles arrivant par UUCP en arrière-plan, il vous faut de bonnes performances du système de fichiers, même sur un système mono-utilisateur. C'est peut-être une optimisation inutile aux yeux du théoricien et un bitouillage de performance (pour reprendre vos propres termes), mais je m'intéresse à un Vrai système d'exploitation - pas à un jouet de chercheur.

Theodore Ts'o bloom-beacon !mit-athena !tytso 308 High St., Medford, MA 02155 tytso@athena.mit.edu Tous jouent au jeu, mais aucun avec les mêmes règles !

De : joe@jshark.rn.com
Sujet : Re : LINUX est dépassé
Date : 31 Jan 92 13 :21 :44 GMT
Organisation : a blip of entropy

Dans l'article <12595@star.cs.vu.nl> ast@cs.vu.nl (Andy Tanenbaum) écrit :

> MINIX a été conçu pour être raisonnablement portable, et a été porté de la ligne Intel aux 680x0 (Atari, Amiga, Macintosh), SPARC, et NS32016. Linux est attaché de très près au 80x86. C'est une mauvaise approche.

Si vous regardiez le source au lieu de croire l'auteur, vous réaliseriez que ce n'est pas vrai !

Il a remplacé « fubyte » par une routine qui utilise explicitement un registre de segment - mais cela peut être facilement modifié. De même, à part à quelques endroits qui présument la présence d'une MMU 386, deux ou trois macros qui cachent les tailles exactes de pages, etc. rendraient triviale la tâche de portage. L'utilisation des TSS 386 simplifient le code, mais le VAX et le WE32000 ont des structures similaires.

Comme il l'a déjà admis, un peu de planification rendrait le système plus propre, mais ce n'est pas un crime de mettre un peu de langage d'assemblage 386 ici ou là !

Et avec tout le respect que je dois :

- le Livre ne fait pas un but de la portabilité (à part sur quelques « #ifdef M8088 ») ;
- au moment de sa sortie, Minix dépendait de certaines « fonctionnalités » du 8086 qui ont causé l'ire des utilisateurs de 68000.

joe.

De : entropy@wintermute.WPI.EDU (Lawrence C. Foard)
Sujet : Re : LINUX est dépassé
Date : 5 Fév 92 14 :56 :30 GMT
Organisation : Worcester Polytechnic Institute

Dans l'article <12595@star.cs.vu.nl> ast@cs.vu.nl (Andy Tanenbaum) écrit :

> Ne le prends pas mal, je ne suis pas mécontent de LINUX. Il va amener tous ceux qui veulent faire évoluer MINIX en BSD UNIX derrière moi. Mais en toute honnêteté, je voudrais suggérer à ceux qui veulent un SE « libre » MODERNE qu'ils recherchent un SE à micro-noyau, portable, comme peut-être GNU ou quelque chose comme ça.

Je crois que certains points de votre position sont valides, bien que je ne sois pas sûr qu'un micro-noyau soit nécessairement meilleur. Il peut y avoir plus de sens de permettre une certaine combinaison des deux. Dans le code d'IPC que je suis en train d'écrire pour Linux, je vais inclure du code qui permettra aux gestionnaires de périphériques et aux systèmes de fichiers tourner en mode utilisateur. Cela sera significativement plus lent malgré tout, et je crois qu'il serait une erreur de tout sortir du noyau (TCP/IP sera interne).

En réalité, mon principal problème avec les théoriciens des SE est qu'ils n'ont jamais testé leurs idées ! Aucune de celles-ci (à l'exception partielle de MACH) n'a jamais vu le jour. Les ordinateurs personnels 32 bits sont disponibles depuis au moins une décennie et Linus a été le premier à écrire un SE qui fonctionne pour eux sans devoir verser 100 000$ à ATT. Un morceau de logiciel en mains vaut mieux que dix logiciels fictifs, les théoriciens des SE sautent vite sur un SE mais ne veulent même pas fournir une autre option.

Le consensus général que les micro-noyaux sont la direction à prendre ne veut rien dire tant qu'aucune application réelle n'a fonctionné sur l'un d'entre eux.

La sortie de Linux me permet d'essayer quelques-unes des idées que je souhaite expérimenter depuis des années, mais je n'avais jamais eu l'occasion de travailler avec le code source d'un SE qui fonctionne.

De : ast@cs.vu.nl (Andy Tanenbaum)
Sujet : Re : LINUX est dépassé
Date : 5 Fév 92 23 :33 :23 GMT
Organisation : Fac. Wiskunde & Informatica, Vrije Universiteit, Amsterdam

Dans l'article <1992Feb5.145630.759@wpi.WPI.EDU>
entropy@wintermute.WPI.EDU (Lawrence C. Foard) écrit :

> En réalité, mon principal problème avec les théoriciens des SE est qu'ils n'ont jamais testé leurs idées !

Je suis mortellement injurié. JE NE SUIS PAS UN THÉORICIEN. Demandez à tous ceux qui étaient à la réunion du département hier (je plaisante).

En fait, ces idées ont été très bien testées en pratique. OSF est en train de jouer son va-tout sur un micro-noyau (Mach 3.0). USL joue son affaire sur un autre (Chorus). Tous deux font tourner de nombreux logiciels et ont été intensivement comparés à des systèmes monolithiques. Amoeba a été complètement implémenté et testé pour un certain nombre d'applications. QNX est bâti autour d'un micro-noyau, et on m'a dit que le parc installé était de 200 000 systèmes. Les micro-noyaux ne sont pas des châteaux en Espagne. Ils reposent sur une approche éprouvée.

Les gars de Mach ont écrit un papier appelé « UNIX comme programme applicatif ». C'était de Golub et al., durant la conférence USENIX de l'été 1990. Les gens de Chorus ont eux aussi un rapport technique sur les performances des micro-noyaux, et je suis co-auteur d'un autre papier sur le sujet, que j'ai mentionné hier (Computing Systems de Déc. 1991). Jetez-y un il.

Andy Tanenbaum (ast@cs.vu.nl)

De : peter@ferranti.com (peter da silva)
Sujet : Re : LINUX est dépassé
Date : Jeu, 6 Fév 1992 16 :02 :47 GMT
Organisation : Xenix Support, FICC

Dans l'article <12747@star.cs.vu.nl> ast@cs.vu.nl (Andy Tanenbaum) écrit :

> QNX est un système à micro-noyau, et on m'a dit que le parc installé était de 200 000 systèmes.

Ah, oui, tant que je suis sur le sujet... il y a plus de 3 millions d'Amiga dans la nature, ce qui veut dire qu'il y en a plus que ce que n'importe quel constructeur d'UNIX a vendu, et probablement plus que tous les systèmes UNIX combinés.

De : peter@ferranti.com (peter da silva)
Sujet : Re : LINUX est dépassé
Date : Xenix Support, FICC
Organisation : Jeu, 6 Fév 1992 16 :00 :22 GMT

Dans l'article <1992Feb5.145630.759@wpi.WPI.EDU>
entropy@wintermute.WPI.EDU (Lawrence C. Foard) écrit :

> En réalité, mon principal problème avec les théoriciens des SE est qu'ils n'ont jamais testé leurs idées !

Je m'inscris en faux... il y a de nombreux systèmes à micro-noyau dans la nature pour tout depuis un 8088 (QNX) jusqu'à de grands systèmes de recherche.

> Aucune de celles-ci (à l'exception partielle de MACH) n'a jamais vu le jour. Les ordinateurs personnels 32 bits sont disponibles depuis au moins une décennie et Linus a été le premier à écrire un SE qui fonctionne pour eux sans devoir verser 100 000$ à ATT.

Je dois avoir rêvé AmigaOS, alors. J'ai utilisé un produit de mon imagination durant les 6 dernières années.

AmigaOS a une conception de passage de messages de micro-noyau, avec un temps de réponse et des performances meilleurs que tout autre système d'exploitation pour PC disponible : y compris MINIX, OS/2, MS-Windows, MacOS, Linux, UNIX, et **certainement** MS-DOS.

La conception micro-noyau a prouvé sa valeur incalculable. Des choses comme de nouveaux systèmes de fichiers disponibles en principe seulement auprès de leur constructeur sont des produits de l'activité de loisirs sur l'Amiga. Les pilotes de périphériques sont simplement des bibliothèques partagées avec des points d'entrée spécifiques et des ports de messages. Il en est de même pour les systèmes de fichiers, le système de fenêtrage, etc. C'est une conception MERVEILLEUSE, qui valide tout ce que les gens ont pu dire sur les micro-noyaux. Oui, cela représente un plus gros travail de les faire décoller qu'un macro-noyau fondé sur les co-routines comme UNIX, mais la souplesse vous le rembourse plusieurs fois.

Je souhaite vraiment qu'Andy fasse un nouveau MINIX fondé sur ce qui a été appris depuis la première sortie. La construction des responsabilités dans MINIX est assez pauvre, mais le concept de base est bon.

> Le consensus général que les micro-noyaux sont la direction à prendre ne veut rien dire tant qu'aucune application réelle n'a fonctionné sur l'un d'entre eux.

Je rêve encore. J'ai sûrement imaginé que Deluxe Paint, Sculpt 3d, Photon Paint, Manx C, Manx SDB, Perfect Sound, Videoscape 3d, et les autres programmes que j'ai achetés étaient « réels ». Je vais avoir à renvoyer ces sacrés trucs, je pense.

La disponibilité de Linux est une excellente chose. Je suis ravi qu'il existe. Je suis sûr que sa conception macro-noyau est une des raisons de son implémentation si rapide, et c'est une raison valable d'utiliser ceux-ci. MAIS... cela ne veut pas dire que les micro-noyaux soient intrinsèquement lents, ni de simples jouets de chercheurs.

De : dsmythe@netcom.COM (Dave Smythe)
Sujet : Re : LINUX est dépassé
Date : 10 Fév 92 07 :08 :22 GMT
Organisation : Netcom - Online Communication Services (408 241-9760 guest)

Dans l'article <1992Feb5.145630.759@wpi.WPI.EDU>
entropy@wintermute.WPI.EDU (Lawrence C. Foard) écrit :

> En réalité, mon principal problème avec les théoriciens des SE est qu'ils n'ont jamais testé leurs idées ! Aucune de celles-ci (à l'exception partielle de MACH) n'a jamais vu le jour.

David Cheriton (Prof. à Stanford, et auteur du systemV) disait quelque chose de similaire dans son cours de systèmes distribués. En paraphrasant :

« Il y a deux types de chercheurs : ceux qui ont implémenté quelque chose et ceux qui ne l'ont pas fait. Ces derniers vous diront qu'il y a 142 manières de faire les choses et qu'il n'y a pas de consensus sur la meilleure. Les premiers vous diront simplement que 141 d'entre elles ne fonctionnent pas. »

Il tire aussi sur les ISO-philes, pour des raisons similaires. Les protocoles Internet ne sont adoptés qu'après avoir été utilisés pendant un certain temps, ce qui évite de standardiser des choses qui ne seront jamais implémentables de manière raisonnable. Les adhérents à l'ISO, d'un autre côté, semblent tenter de standardiser tout ce qui est possible, y compris les « évasions » du standard, avant qu'une implémentation raisonnable de référence existe. En conséquence, on trouve immortalisées des idées dépassées, telles que le paquet de champs de données de niveau infra-octet, ce qui rend difficile de réaliser de bonnes performances lorsque votre ordinateur est alimenté par une lance d'incendie à plus de 10 Gbit/s :-)

Juste mes 0,02$

D

De : torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Sujet : Excuses (était Re : LINUX est dépassé)
Date : 30 Jan 92 15 :38 :16 GMT
Organisation : University of Helsinki

Dans l'article <1992Jan29.231426.20469@klaava.Helsinki.FI> j'ai écrit :

> Bien, avec un sujet tel que celui-ci, j'ai bien peur de devoir répondre.

Et j'ai répondu, je me suis laissé aller, sans une pensée pour le bon goût et la netiquette. Pardon à ast, et merci à John Nall pour sa lettre « ce n'est pas comme ça que l'on fait ». Je me suis emporté, et je vais composer maintenant une lettre personnelle (beaucoup moins acerbe) à ast. J'espère que personne ne se détournera de linux parce qu'il est (a) peut-être dépassé (je pense toujours que ce n'est pas le cas, bien que certaines critiques soient valables) et (b) qu'il a été écrit par une tête brûlée :-)

Linus « mon premier, et j'espère mon dernier festival incendiaire » Torvalds

De : pmacdona@sanjuan (Peter MacDonald) Sujet : Re : Linux est dépassé Date : 1 Fév 92 02 :10 :06 GMT Organisation : University of Victoria, Victoria, BC, CANADA

Depuis, je pense, que j'ai posté l'un des messages précédents dans toute cette discussion sur Linux contre Minix, je me sens obligé de commenter mes raisons de basculer de Minix vers Linux. Par ordre d'importance, elles sont :

1) Linux est gratuit ;
2) Linux évolue de manière satisfaisante (car les nouvelles fonctionnalités sont acceptées dans la distribution par Linus).

La première nécessite quelques explications, puisque si j'ai déjà acheté Minix, en quoi le prix peut-il me concerner ? Simple. Si le système d'exploitation est gratuit, beaucoup plus de gens vont l'utiliser/le maintenir/l'améliorer. C'est aussi le raisonnement que j'ai tenu lorsque j'ai acheté mon 386 au lieu d'une SPARC (que j'aurais pu avoir pour seulement 30% de plus). Depuis que les PC sont bon marché et partout disponibles, de plus en plus de gens les achètent/les utilisent et ainsi, le logiciel bon marché/gratuit est abondant.

La seconde doit être assez évidente pour quiconque a utilisé Minix durant un certain temps. AST n'accepte généralement pas d'améliorations à Minix. Ce n'est pas destiné à être un challenge, mais seulement un fait. AST a de bonnes et légitimes raisons pour cela, et je ne les discute pas. Mais Minix a quelques limitations avec lesquelles je ne peux plus continuer à vivre, et à cause de cette position, l'hypothèse de les voir résolues dans un temps raisonnable n'est pas satisfaisante. Parmi ces limitations, il y a :

- pas de gestion des instructions 386 ;
- pas de console virtuelle ;
- pas de liens symboliques ;
- pas d'appel select ;
- pas de pseudo-terminal ;
- pas de pagination à la demande/pagination disque/ texte partagé/bibliothèque partagée (mm efficace)... chmem (mm inflexible) ;
- pas de X-Window (pour les mêmes raisons que celles invoquées pour Linux et les 386) ;
- pas de TCP/IP ;
- pas d'intégration GNU/SysV (portabilité) ;
- ...

Certaines de ces limitations peuvent être levées par des correctifs (et si vous l'avez réalisé vous-même, je n'ai pas besoin de vous dire la satisfaction que cela procure), mais au moins les cinq derniers points étaient/sont hors de toute attente raisonnable.

Finalement, mon commentaire (ma saillie ?) sur l'architecture en noyau segmenté, ou micro-noyau de Minix était plus une expression de ma frustration/désorientation dans ma tentative d'utilisation des correctifs de pseudo-terminaux de Minix comme un guide sur la manière de le réaliser sous Linux. Cette fonction particulière était une de celles pour lesquelles le passage de messages en rendait l'implémentation vraiment complexe.

J'ai une opinion sur le sujet Monolithique contre passage de messages, mais je ne vais pas l'exprimer maintenant, et je n'ai pas songé à le faire jusqu'à présent. Mes buts sont totalement à court terme (fonctionnalités maximales et temps/coût/effort minimaux), c'est pourquoi mes vues ne sont pas appropriées, et ne doivent pas être mal interprétées. Si le manque des fonctionnalités ci-dessus ne vous gêne pas, alors vous devez prendre Minix en considération (si cela ne vous gêne pas de payer bien sûr :).

De : olaf@oski.toppoint.de (Olaf Schlueter)
Sujet : Re : Linux est dépassé
Date : 7 Fév 92 11 :41 :44 GMT
Organisation : Toppoint Mailbox e.V.

Juste quelques commentaires sur la discussion Linux contre Minix, qui devient en partie une discussion monolithique contre micro-noyau.

Je pense qu'il n'y aura pas d'accords entre les tenants de chacun de ces concepts, s'ils oublient que Linux et Minix ont été conçus pour des utilisations différentes. Si vous voulez un système Unix abordable, puissant et évolutif sur une machine unique, avec la possibilité d'adapter des logiciels Unix sans difficulté, alors Linux est fait pour vous. Si vous vous intéressez aux concepts des systèmes d'exploitation modernes, et que vous voulez apprendre comment fonctionne un micro-noyau, alors Minix est

le meilleur choix.

Ce n'est pas un argument contre le système micro-noyau, que pour l'instant les implémentations monolithiques d'Unix sur PC aient de meilleures performances. Cela veut simplement dire qu'Unix est peut-être mieux implémenté en monolithique, au moins tant qu'il s'exécute sur une seule machine. Du point de vue des utilisateurs, la conception interne du système n'a aucune importance. Jusqu'à ce qu'il soit en réseau. Selon l'approche monolithique, un serveur de fichiers deviendra un processus utilisateur qui s'appuie sur un matériel quelconque comme Ethernet. Les programmes qui souhaitent utiliser ce support devront utiliser des bibliothèques spécifiques qui offrent les appels de communication avec ce serveur. Dans un système micro-noyau il est possible d'incorporer tout le serveur dans le système sans qu'il soit besoin de « nouveaux » appels système. Du point de vue de l'utilisateur c'est un avantage, car rien ne change, il bénéficie juste de meilleures performances (en termes d'augmentation d'espace disque, par exemple). Du point de vue de l'implémenteur, le micro-noyau est plus rapidement adaptable à des modifications de la conception matérielle.

Il a été critiqué qu'AST rejette toute amélioration à Minix. Puisque son intérêt est dans la valeur éducative de Minix, je comprends cet argument de vouloir conserver la simplicité du code, et de ne pas vouloir le surcharger de fonctionnalités. En tant qu'outil éducatif, Minix est écrit comme un système micro-noyau, bien qu'il fonctionne sur des plates-formes qui se comporteraient probablement mieux avec un système monolithique. Mais la zone d'influence des applications réseau croît et des systèmes modernes comme Amoeba ou Plan 9 ne peuvent être écrits sous forme monolithique. Ainsi Minix a-t-il été écrit dans l'intention de donner aux étudiants un exemple pratique d'un système micro-noyau, pour qu'ils puissent jouer avec les tâches et les messages. Ce n'était pas l'idée de fournir à de nombreuses personnes un système économique et puissant pour le dixième du prix d'implémentations SYSV ou BSD.

Résumé : Linux n'est pas meilleur que Minix, ni le contraire. Ils sont différents pour de bonnes raisons.

De : meggin@epas.utoronto.ca (David Megginson)
Sujet : Mach/Minix/Linux/Gnu etc.
Date : 1 Fév 92 17 :11 :03 GMT
Organisation : University of Toronto - EPAS

Bien, c'était une chouette discussion. Je suis totalement convaincu par le professeur Tanenbaum qu'un micro-noyau *est* la direction à prendre, mais plus je regarde dans le source de Minix, moins je crois que ce soit un micro-noyau. Je ne m'occuperai sans doute pas du portage de Linux sur le M68000, mais je souhaite plus de services que Minix puisse offrir.

Quid d'un micro-noyau compatible MACH au niveau message/appel système ? Il n'a pas en fait besoin de faire tout ce que fait MACH, comme la pagination virtuelle de mémoire, il doit juste *avoir l'air* de MACH depuis l'extérieur, pour tromper des programmes comme le futur émulateur Unix Gnu, BSD, etc. Cela prolongerait un peu la vie utile de nos machines sur M68000 ou sur 286. Dans l'intervalle, je vais probablement rester sous Minix avec mon ST plutôt que repasser à MiNT après tout, Minix au moins ressemble à Unix, alors que MiNT ressemble au TOS en essayant de ressembler à Unix (il est bien obligé d'être compatible TOS).

David

De : peter@ferranti.com (peter da silva)
Groupe de discussion : comp.os.minix
Sujet : Que rapporte cette guerre ? (Re : LINUX est dépassé)
Date : 3 Fév 92 16 :37 :24 GMT
Organisation : Xenix Support, FICC

Allez-vous arrêter de vous descendre en flamme les uns les autres ? Je veux dire par là que linux est conçu pour fournir un environnement ayant d'assez bonnes performances sur un matériel handicapé par des années passées à traîner le boulet de la compatibilité ascendante. Minix est conçu comme outil d'enseignement. Il n'est pas bon de faire le travail d'un autre, et pourquoi le ferait-il ? Le fait que Minix s'essouffle rapidement (et c'est vrai) n'est pas un problème dans son milieu de prédilection. Il est sûrement meilleur que le système d'exploitation JOUET. Le fait que Linux ne soit pas transposable sur d'autres plates-formes que le 386/AT ne pose pas problème car on en trouve des millions ici-bas (et vraiment pas chers : vous pouvez trouver un 386/SX en dessous de 1000 dollars).

Un noyau monolithique est assez facile à fabriquer, ce qui vaut le coup si cela permet au système de sortir rapidement. Considérez-le comme une économie de travail pour le temps du programmeur. L'API est portable. Vous pouvez remplacer le noyau par un autre ayant la conception micro-noyau (et en matière de conception de micro-noyau MINIX ne représente pas le top du top, même pour des PC d'entrée de gamme... voyez l'AmigaOS) sans perturber les applications. C'est tout l'intérêt d'avoir une API portable au départ.

Les micro-noyaux sont certainement ce qu'il y a de mieux pour beaucoup de choses. Cela demande plus de travail pour en améliorer l'efficacité, et dès lors une conception plus simple, ne tirant réellement pas partie des avantages du micro-noyau est préférable, si on veut le faire pour des raisons pédagogiques. Pensez que c'est une économie de temps pour les étudiants. Le concept est toujours bon et lorsqu'il est possible d'avoir une interface de programmation d'applications pour le micro-noyau on peut obtenir une amélioration TRÈS impressionnante (des milliers de commutations par seconde avec un 68000 8 MHz).

De : ast@cs.vu.nl (Andy Tanenbaum)
Sujet : Campeurs mécontents
Date : 3 Fév 92 22 :46 :40 GMT
Organisation : Fac. Wiskunde & Informatica, Vrije Universiteit, Amsterdam

J'ai récemment reçu quelques courriers de campeurs mécontents (en fait 10 messages venant des 43000 lecteurs pourraient paraître beaucoup, mais ça ne l'est pas réellement). Il semble en ressortir trois points :

1. Les noyaux monolithiques sont tout aussi bien que les micro-noyaux
2. La portabilité n'est pas si importante
3. Les logiciels devraient être gratuits

Si quelqu'un veut avoir une discussion sérieuse entre micro-noyaux et noyaux monolitiques, très bien. Nous pouvons le faire sur comp.os.research. Mais par pitié ne venez pas faire de bruit si vous n'avez aucune idée de ce dont vous parlez. J'ai aidé à concevoir et implémenter 3 systèmes d'exploitation, un monolithique et deux micro, et étudié beaucoup d'autres en détail. Beaucoup des arguments proposés ne valent rien (par exemple, les micro-noyaux ne sont pas bons car on ne peut pas

faire de pagination au niveau utilisateur - alors que Mach FAIT de la pagination au niveau utilisateur).

Si vous ne connaissez pas grand-chose à propos de la discussion micro-noyaux contre noyaux monolithiques, il existe quelques informations utiles dans un article que j'ai écrit en collaboration avec Fred Ouglis, Frans Kaashoek et John Ousterhout dans le numéro de Déc. 1991 de COMPUTING SYSTEMS, le journal USENIX). Si vous n'avez pas ce journal, vous pouvez FTPer l'article depuis ftp.cs.vu.nl (192.31.231.42) dans le répertoire amoeba/papers sous comp_sys.tex.Z (source TeX compacté) ou comp_sys.ps.Z (PostScript compacté). L'article donne des mesures de performances actuelles et appuie la conclusion de Rick Rashid comme quoi les systèmes à micro-noyaux sont tout aussi efficaces que les noyaux monolithiques.

À propos de la portabilité, il n'y a plus de discussion possible. UNIX a été porté sur tout depuis les PCs jusqu'aux Crays. Écrire un SE portable n'est pas beaucoup plus dur qu'un non portable, et tous les systèmes doivent être écrits en gardant à l'esprit la portabilité de nos jours. Le professeur de SE de Linus l'a sûrement signalé. Faire un SE portable n'est pas quelque chose que j'ai inventé en 1987.

Alors que l'on peut rationnellement parler de la conception du noyau et de la portabilité, la question de la gratuité est 100% émotionnelle. Vous ne pourrez pas croire à quel point j'ai été dernièrement [juron supprimé] au sujet de la non gratuité de MINIX. MINIX coûte 169$, mais la licence permet de faire deux copies de sauvegarde, le prix effectif peut donc être de moins de 60$. De plus, les professeurs peuvent en faire des copies ILLIMITÉES pour leurs étudiants. Coherent coûte 99$. FSF facture plus de 100$ pour les bandes sur lesquelles sont fournies son logiciel « gratuit » si vous n'avez pas d'accès à l'Internet, et je n'ai jamais entendu personne se plaindre. 4.4 BSD coûte 800$. Je ne pense vraiment pas que l'argent soit la question. D'ailleurs, il est probable que la plupart des lecteurs de ce forum l'ont déjà.

Une opinion, que je pense tout le monde ne partage pas, est que de rendre quelque chose disponible par FTP n'est pas nécessairement le moyen de fournir la plus large distribution. L'Internet est toujours un groupe hautement élitiste. La plupart des utilisateurs d'ordinateurs n'y sont PAS. J'ai compris d'après PH que le pays où MINIX est le plus utilisé est l'Allemagne, et non les USA, principalement parce qu'un des magazines informatiques (commercial) allemand l'a activement appuyé. MINIX est aussi très utilisé en Europe de l'Est, au Japon, Israël, Amérique du Sud, etc. La plupart de ces gens ne l'auraient jamais obtenu s'il n'y avait eu une entreprise pour le vendre.

Pour revenir à ce que « libre » veut dire, qu'en est-il du code source libre ? Coherent est binaire seulement, mais MINIX a le code source, exactement comme LINUX. Vous pouvez le changer comme vous le désirez, et poster ici les changements. Tout le monde l'a fait pendant 5 ans sans problèmes. J'ai aussi donné des mises à jours gratuites pendant des années.

Je pense que la question réelle est quelque chose d'autre. Je me suis vu offrir de façon répétée mémoire virtuelle, pagination, liens symboliques, systèmes de fenêtrage, et toutes sortes de fonctionnalités. J'ai en général refusé car j'essaie toujours de garder le système assez simple pour qu'il soit compris par des étudiants. Vous pouvez mettre toutes ces choses dans votre version, mais je ne les mettrais pas dans la mienne. Je pense que c'est ce point qui ennuie les gens qui disent que « MINIX n'est pas libre », pas les 60$.

Une question intéressante est de savoir si Linus est prêt à laisser LINUX devenir « libre » de son contrôle. Quelqu'un peut-il le modifier (le ruiner ?) et le vendre ?

Souvenez-vous des centaines de messages avec le sujet « Re : Votre logiciel vendu pour de l'argent » lorsqu'il a été découvert que le centre MINIX en Angleterre vendait des disquettes avec des articles de news, plus ou moins au prix coûtant ?

Supposez que Fred van Kempen revienne et veuille prendre le relais, en créant un LINUX Fred et un LINUX Linus, tous deux utiles mais différents. Est-ce ok ? La question se pose lorsqu'un groupe assez grand de personnes veut faire évoluer LINUX d'une manière que Linus ne veut pas. Avant que ça n'arrive, la question reste néanmoins sur le tapis.

Si vous préférez la philosophie de Linus à la mienne, bien sûr, suivez-le, mais par pitié ne prétendez pas que vous faites ça parce que Linux est « libre ». Dites simplement que vous voulez un système avec un tas de fonctionnalités. Très bien. C'est votre choix. Je n'ai pas d'argument contre ça. Dites juste la vérité.

En aparté, pour ces gens qui ne lisent pas les en-têtes des news, Linus est en Finlande et je suis aux Pays-Bas. Atteignons-nous une situation où une autre industrie critique, le logiciel libre, qui a été totalement dominée par les USA est en train de céder la place à la concurrence étrangère ? Verrons-nous bientôt arriver le Président Bush en Europe avec Richard Stallman et Rick Rashid à la remorque, demandant que l'Europe importe plus de logiciels libres Américains ?

Andy Tanenbaum (ast@cs.vu.nl)

De : ast@cs.vu.nl (Andy Tanenbaum)
Sujet : Re : Campeurs mécontents
Date : 5 Fév 92 23 :23 :26 GMT
Organisation : Fac. Wiskunde & Informatica, Vrije Universiteit, Amsterdam

Dans l'article <205@fishpond.uucp> fnf@fishpond.uucp (Fred Fish) écrit :

> Si PH ne s'était pas octroyé le monopole de la distribution, il aurait été possible pour tous les bitouilleurs minix intéressés d'organiser et de mettre en place un groupe qui se serait consacré à produire un minix amélioré. Le but de ce groupe aurait été de produire une unique version vivante de minix avec toutes les améliorations couramment demandées. Cela aurait permis à minix d'évoluer un peu de la même manière que gcc a évolué durant ces dernières années.

Ceci EST possible. Si un groupe de personnes veut le faire, c'est très bien. Je pense que coordonner 1000 prima donnas vivant tout autour du monde sera aussi facile que de mener des chats en troupeau, mais il n'y a pas de problème légal. Lorsqu'une nouvelle version est prête, il suffit de faire un fichier diff par rapport à la 1.5 et le poster ou le rendre disponible par FTP. Bien que cela requiert quelque travail de la part des utilisateurs pour l'installer, ce n'est pas tant de travail. D'ailleurs, j'ai des scripts shell pour créer les diffs et les installer. C'est ce que Fred van Kempen faisait. Ce qu'il a fait de mauvais est d'insister sur le droit de publier la nouvelle version, plutôt que les diffs par rapport à la base de PH. Ceci mettait PH hors course, ce qui, rien d'étonnant, ne les emballait pas. Si des personnes veulent toujours le faire, qu'elles le fassent.

Bien sûr, je ne vais pas nécessairement appliquer tous ces changements à ma version, il y a donc du travail pour garder la version officielle et celles qui sont améliorées en synchro, mais je suis disposé à coopérer pour minimiser le travail. Je l'ai fait pendant longtemps avec Bruce Evans et Frans Meulenbroeks.

Si Linus veut garder le contrôle de la version officielle, et qu'un groupe d'acharnés veulent continuer dans une direction différente, le même problème arrive. Je ne pense pas que le droit de copie soit réellement le problème. Le problème est de coordonner les choses. Des projets comme GNU, MINIX, ou LINUX tiennent debout seulement si une personne en a la charge. Durant les années 1970, lorsque la programmation structurée a été introduite, Harlan Mills signala que l'équipe de programmation devait être organisée comme une équipe chirurgicale (un chirurgien et ses assistants), et non comme une équipe de bouchers (donnez à chacun un morceau et laissez-les hacher seuls).

Quiconque disant que l'on peut avoir un grand nombre de personnes très dispersés bitouillant sur une même partie complexe de code et éviter l'anarchie totale n'a jamais dirigé un projet logiciel.

> Où est le groupe important de personnes désirant fair évoluer gcc d'une manière que rms/FSF n'approuve pas ?

Un compilateur n'est pas une chose pour laquelle les gens ont un grand attachement émotionnel. Si le langage à compiler est une donnée (par exemple, une norme ANSI), il n'y a pas de place pour que quiconque invente de nouvelles fonctionnalités. Un système d'exploitation a des occasions illimitées pour que des personnes implémentent leurs fonctionnalités préférées.

Andy Tanenbaum (ast@cs.vu.nl)

De : torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Sujet : Re : Campeurs mécontents
Date : 6 Fév 92 10 :33 :31 GMT
Organisation : University of Helsinki

Dans l'article <12746@star.cs.vu.nl> ast@cs.vu.nl (Andy Tanenbaum) écrit :

> Si Linus veut garder le contrôle de la version officielle, et qu'un groupe d'acharnés veulent continuer dans une direction différente, le même problème arrive.

C'est la seconde fois que je vois cette « accusation » de la part de ast, qui trouve malin de faire des commentaires sur un noyau qu'il n'a probablement jamais vu. Ou tout au moins il ne m'a jamais questionné, ou bien lu alt.os.unix sur le sujet. Seulement pour que personne ne prenne ses suppositions comme vérité première, voici ma position sur « la prise de contrôle », en 2 mots (ou trois ?) :

J'ferai rien.

Le seul contrôle que je garde effectivement sur linux est que je le connais mieux que quiconque, et que j'ai fait en sorte que mes changements soient disponibles sur les sites ftp, etc. Ceux-ci sont devenus des versions officielles et je ne m'attends pas à ce que cela change pour quelque temps : non pas parce que j'estime que j'ai un droit moral sur lui, mais je n'ai pas entendu trop de réclamations et il se passera beaucoup de mois avant que je ne puisse trouver des gens ayant le même « feeling » au sujet du noyau. (Bon, peut-être certains y arrivent : tytso a effectué assurément des changements significatifs même jusqu'au 0.10, et d'autres l'ont bitouillé également).

Dans les faits, j'ai lancé quelques ballons d'essai sur une liste de diffusion « noyau-linux » qui prendraient les décisions sur les mises à jour, et je m'attends à ce que je ne puisse pas apporter toutes les fonctionnalités qui *devront* être ajoutées : SCSI

etc, car je ne dispose pas du matériel. Le résultat fut négatif : les gens ne semblent pas pressés de changer à l'heure actuelle. (oui, quelqu'un a pensé que je devrais demander aux alentours des dons pour pouvoir développer le logiciel nécessaire - et si quelqu'un a du matériel intéressant, je serai heureux de l'accepter :)

La seule chose que le copyright interdise (et je pense que cela est éminemment raisonnable) est que d'autres personnes commencent à se faire de l'argent avec et ne fournissent pas les sources, etc. Ce n'est pas une question de logique, mais je le vivrais mal si quelqu'un venait à vendre mon travail pour de l'argent, alors que je l'avais rendu disponible exprès afin que tout le monde puisse s'amuser comme il l'entend. Je pense que la plupart des gens comprennent mon point de vue.

Ceci mis à part, si Fred van Kempen veut faire un super-linux, il est tout-à-fait le bienvenu. Il ne se fera pas beaucoup d'argent avec (hormis les frais de distribution), et je ne pense pas que ce soit une bonne idée de diviser linux, mais je ne voudrais pas arrêter même si le copyright me le permet.

> Je ne pense pas que le droit de copie soit réellement le problème. Le problème est de coordonner les choses. Des projets comme GNU, MINIX, ou LINUX tiennent debout seulement si une personne en a la charge.

Oui, la coordination est un gros problème, et je ne pense pas que je quitterai le poste de « chirurgien chef » pour linux avant longtemps, en partie parce que la plupart des gens comprennent ces problèmes. Mais le copyright *est* une solution : si on sent que je fais du mauvais boulot, ils peuvent le faire eux-mêmes. Tout comme pour gcc. Le copyright de minix, cependant, signifie que si quelqu'un a le sentiment qu'il peut faire un meilleur minix, soit il doit faire des patches (qui ne sont pas si gros quoi que vous ayez dit à ce sujet) ou démarrer à partir de rien (et être attaqué car vous avez d'autres idéaux).

Les patches ne sont pas très drôles à diffuser : je n'ai pas encore fait de cdiff pour une seule version de linux (j'espère que ça changera : bientôt les patches seront beaucoup plus petits que le noyau et faire à la fois les patches et une version complète deviendra une bonne idée - notez que j'ai toujours donné aussi la version complète). Faire des patches sur des patches est simplement impraticable, spécialement pour les personnes pouvant effectuer les changements elles-mêmes.

> Où est le groupe important de personnes désirant faire évoluer gcc d'une manière que rms/FSF n'approuve pas ?
>
> Un compilateur n'est pas une chose pour laquelle les gens ont un grand attachement émotionnel. Si le langage à compiler est une donnée (par exemple, une norme ANSI), il n'y a pas de place pour que quiconque invente de nouvelles fonctionnalités. Un système d'exploitation a des occasions illimitées pour que des personnes implémentent leurs fonctionnalités préférées.

Bien, il y a GNU emacs... Ne nous dites pas que les gens n'ont pas de relation émotionnelle avec leurs éditeurs :)

Linus

De : dmiller@acg.uucp (David Miller)
Sujet : Linux est dépassé et messages suivants
Date : 3 Fév 92 01 :03 :46 GMT
Organisation : AppliedComputerGroup

En tant qu'observateur intéressé dans la conception de systèmes, je ne peux résister à cette enfilade. Rendez-vous compte que je ne suis pas réellement expérimenté avec minux [1] ni linux. Je suis avec unix depuis de nombreuses années. Tout d'abord, quelques observations :

Minix fut écrit pour être un outil éducatif pour les classes d'AST, pas un système d'exploitation commercial. Il n'a jamais été un paramètre de conception de le faire fonctionner en code source librement disponible pour les systèmes unix. Je pense qu'il était aussi une présentation de la manière dont les systèmes d'exploitation doivent être conçus, avec un micro-noyau et des processus séparés couvrant autant de fonctionnalités que possible.

Linux fut écrit essentiellement en tant qu'exercice d'apprentissage de la part de Linus comment programmer la famille 386. La conception du système d'exploitation ultime n'était pas un objectif. Fournir une plate-forme utilisable, libre, qui puisse exécuter toutes sortes de logiciels libres partout disponibles fut une considération, et c'est une de celles qui apparaît avoir été remplie au mieux.

La critique de qui que ce soit considérant que l'un de ces systèmes n'est pas ce que **lui** aurait voulu qu'il soit est déplacée. Après tout, quiconque possédant un ordinateur capable d'exécuter l'un ou l'autre de ces systèmes est libre de réaliser lui-même ce que Linus et Andrew ont écrit !

Moi, pour ce que je suis, j'applaudis Linus pour son effort considérable dans le développement de Linux et dans sa décision de le rendre libre pour tous. J'applaudis AST pour son effort pour rendre minix accessible. J'ai un gros problème par rapport aux remarques sur l'aspect non-libre de minix. Si vous pouvez trouver le temps d'explorer minix et un système d'ordinateur de base, 150 dollars ne sont pas grand-chose et vous avez en plus un livre pour l'accompagner.

Ensuite, quelques questions pour le professeur :

Minix est-il supposé être un « vrai système d'exploitation » ou un outil éducatif ? En tant qu'outil éducatif, c'est un excellent travail. En tant que vrai système d'exploitation, il présente quelques manques terriblement gênants (pourquoi pas de malloc() ?, juste pour les débutants). Mon impression à la lecture du Livre et des messages qui passent ici est que vous vouliez un outil pour enseigner en classe, et que de nombreux autres souhaitaient jouer avec un système d'exploitation accessible. Ces autres ont tenté de lui greffer suffisamment de fonctions pour en faire un « vrai système d'exploitation », avec un succès moins éclatant.

Pourquoi séparer les fonctions fondamentales du SE, telles que la gestion mémoire, dans des processus utilisateurs ? Comme le savent tous les bons gourous *nix, la voie du succès est de diviser pour régner, avec pour but de **simplifier** le problème en composants gérables, bien définis. Si le découpage des éléments de base du système d'exploitation en processus de l'espace utilisateur complique la fonction en y introduisant des mécanismes additionnels (passage de messages, signaux compliqués), avons-nous atteint l'objectif de simplifier la conception et l'implémentation ?

Je suis d'accord que *nix a souffert d'un mauvais niveau de fonctionnalités et particulièrement sysVr4. Peut-être les fonctions que les gens veulent pour telle ou telle fonctionnalité ou compatibilité peuvent-elles être offertes par des modules/bibliothèques chargeables dynamiquement ? Le micro-noyau serait toujours un gestionnaire de ressources de bas niveau capable de router les demandes de fonctions au module ou à la bibliothèque approprié. Les modules peuvent être des threads ou des processus utilisateurs (je pense. Bitouilleurs de systèmes, corrigez-moi :-)).

1. mot utilisé dans la VO

Juste la valeur de mes 0,04 dollars. N'hésitez pas à m'envoyer vos réponses ici ou par la messagerie électronique. Je n'ai pas de formation traditionnelle en informatique, et je pose donc réellement ces questions naïvement. Je suppose qu'un tas d'autres sur le Net ont des questions similaires en tête, mais je me suis déjà trompé.

– David

De : michael@gandalf.informatik.rwth-aachen.de (Michael Haardt)
Sujet : 1.6.17 résumé et pourquoi je pense qu'AST a raison.
Date : 6 Fév 92 20 :07 :25 GMT
Répondre à : u31b3hs@messua.informatik.rwth-aachen.de (Michael Haardt)
Organisation : Gandalf - a 386-20 machine

Il y a quelques temps, j'ai demandé des détails sur la prochaine version de MINIX (1.6.17). J'ai obtenu quelques réponses, mais seulement de gens utilisant la 1.6.16. Les informations qui suivent sont officieuses et peuvent être erronées, mais elles sont tout ce que j'en sais actuellement. Corrigez-moi si quelque chose est faux.

– Les correctifs 1.6.17 s'appliqueront sur la 1.5 fournie par PH ;
– Les fichiers d'en-tête sont propres ;
– Les deux types de systèmes de fichiers peuvent être utilisés simultanément ;
– La gestion des signaux est réécrite pour POSIX. Le vieux bogue est supprimé ;
– Le compilateur ANSI (disponible auprès de Transmediar, je crois) est fourni avec les binaires du compilateur et de nouvelles bibliothèques ;
– Le protocole réseau d'Amoeba ne semble pas pris en charge ;
– times(2) renvoit une valeur correcte. termios(2) est implémenté, mais c'est surtout une bitouille. Je ne sais pas si « implémenté » veut dire dans le noyau ou l'émulation actuelle ;
– Le nouveau système de fichiers n'est pas documenté. Il y a de nouveaux fsck et mkfs, pas d'infos sur cela ;
– Le compilateur ANSI travaille mieux sur les flottants ;
– L'ordonnanceur est amélioré, mais pas aussi bien écrit que celui de Kai-Uwe Bloem.

J'ai demandé cela pour appuyer sur des faits ma décision de migrer vers MINIX 1.6.17 ou vers Linux après la fin des examens. Eh bien celle que j'ai prise est de migrer vers Linux à la fin du mois et de retirer MINIX de mon disque lorsque Linux fera tourner tous les logiciels dont j'ai besoin qui tournent actuellement sous MINIX 1.5 avec des correctifs lourds. J'imagine que cela pourra prendre deux mois au plus. Voici les principales raisons de ma décision :

– Il n'y a pas de version « en cours » de MINIX qui puisse être utilisée comme base de correctifs, et personne ne connaît la date d'arrivée de la 1.6.17 ;
– La bibliothèque contient un certain nombre de bogues, et d'après ce que j'ai entendu, ils ne sont pas en cours de correction. Il n'y aura pas de nouveau compilateur, et les utilisateurs en 16 bits continuent à devoir utiliser les ACK bogués ;
– 1.6.17 doit se rapprocher de POSIX, mais il manque encore un termios complet ;
– Je doute qu'il y ait encore beaucoup de développements pour les utilisateurs en 16 bits.

Je pense cesser de maintenir la liste des logiciels MINIX dans quelques mois. Y a-t-il quelqu'un par ici qui veuille le continuer ? Jusqu'à ce que Linux tourne

parfaitement sur ma machine, toutes les mises à jour d'Origami continueront à fonctionner en MINIX 16 bits. J'annoncerai lorsque la dernière de ces versions apparaîtra.

À mon avis, AST a raison dans sa décision concernant MINIX. J'ai lu la guerre incendiaire et ne peux m'empêcher de dire que j'aime MINIX comme il est, par rapport à l'état actuel de Linux. MINIX a quelques avantages :

- Vous pouvez commencer à jouer avec sans disque dur, et même compiler des programmes. Je l'ai fait il y a quelques années ;
- Il est si petit qu'il n'est pas nécessaire d'en savoir beaucoup pour avoir un petit système qui fonctionne correctement ;
- Il y a le livre. Ok, seulement pour la version 1.3, mais le plus gros est toujours valide ;
- MINIX est un exemple de noyau non monolithique. Appelez-le micro-noyau ou bitouille pour contourner une architecture débile : Il démontre un concept, avec ses pour et ses contre - un concept documenté.

Pour moi, c'est un bon système pour les premiers pas dans UNIX et la programmation système. J'ai appris l'essentiel de ce que je sais sur UNIX avec MINIX, dans tous les domaines, depuis la programmation en C sous UNIX jusqu'à l'administration système (et les trous de sécurité). MINIX a grandi avec moi : mises à jour 1.5.xx, consoles virtuelles, courrier et nouvelles, traitement de texte, compilation croisée, etc. Maintenant, il est trop petit pour moi. Je n'ai plus besoin d'un système pour l'enseignement, je voudrais un UNIX plus complexe et plus complet, et il y en a un : Linux.

À l'époque, v7 était l' « état de l'art ». Il y avait MINIX qui en offrait le plus gros. En un ou deux ans, POSIX est devenu ce qu'on a l'habitude de voir. Espérons qu'il y aura MINIX, offrant le plus gros, avec un nouveau livre, pour ceux qui veulent un petit système pour jouer et expérimenter.

Cessez de vous incendier, MINIX et Linux sont deux systèmes différents avec des buts différents. L'un est un outil d'enseignement (et un bon, je pense), l'autre est un vrai UNIX pour les vrais bitouilleurs.

Michael

De : dingbat@diku.dk (Niels Skov Olsen)
Sujet : Re : 1.6.17 résumé et pourquoi je pense qu'AST a raison.
Date : 10 Fév 92 17 :33 :39 GMT
Organisation : Department of Computer Science, U of Copenhagen

michael@gandalf.informatik.rwth-aachen.de (Michael Haardt) écrit :

> Cessez de vous incendier, MINIX et Linux sont deux systèmes différents avec des buts différents. L'un est un outil d'enseignement (et un bon, je pense), l'autre est un vrai UNIX pour les vrais bitouilleurs.

Oyez, oyez ! Et maintenant, les articles sur Linux sont dans alt.os.linux (ou comp.os.misc si vous ne recevez pas la hiérarchie alt.*) et ceux sur Minix ici.

fin (festival d'incendies noyé :-)

Niels

Annexe B

Version 1.0 de la définition de l'Open Source de Bruce Perens

B.1 Version 1.0 de la définition d'OpenSource

OpenSource implique plus que la simple diffusion du code source. La licence d'un programme « open source » doit correspondre aux critères suivants :

B.1.1 Libre redistribution

La licence ne doit pas interdire de vendre ou de donner le logiciel en tant que composant d'une distribution d'un ensemble contenant des programmes de diverses origines. La licence ne doit pas soumettre cette vente à l'acquittement de droits, quels qu'ils soient.

B.1.2 Code source

Le programme doit inclure le code source, et la distribution sous forme de code source comme sous forme compilée doit être autorisée. Quand un produit n'est pas distribué avec le code source correspondant, il doit exister un moyen clairement indiqué de télécharger ce code source, depuis l'Internet, sans frais supplémentaires. Le code source est la forme la plus adéquate pour qu'un programmeur modifie le programme. Il n'est pas autorisé de proposer un code source rendu difficile à comprendre. Il n'est pas autorisé de proposer des formes intermédiaires, comme ce qu'engendre un préprocesseur ou un traducteur automatique.

B.1.3 Travaux dérivés

La licence doit autoriser les modifications et les travaux dérivés, et leur distribution sous les mêmes conditions que celles qu'autorise la licence du programme original.

B.1.4 Intégrité du code source de l'auteur

La licence ne peut restreindre la redistribution du code source sous forme modifiée que si elle autorise la distribution de fichiers « patch » aux côtés du code source dans le but de modifier le programme lors de sa construction.

La licence doit explicitement permettre la distribution de logiciel construit à partir du code source modifié. Elle peut exiger que les travaux dérivés portent un nom différent ou un numéro de version distinct de ceux du logiciel original.

B.1.5 Pas de discrimination sur l'identité de l'utilisateur

La licence ne doit opérer aucune discrimination à l'encontre de personnes ou de groupes de personnes.

B.1.6 Pas de discrimination sur le domaine d'application

La licence ne doit pas limiter le champ d'application du programme. Par exemple, elle ne doit pas interdire son utilisation dans le monde des affaires ou dans le cadre de la recherche génétique.

B.1.7 Distribution de la licence

Les droits attachés au programme doivent s'appliquer à tous ceux à qui le programme est transmis sans que ces parties doivent remplir les conditions d'une licence supplémentaire.

B.1.8 Pas de spécificité

Les droits attachés au programme ne doivent pas dépendre de l'intégration du programme à une distribution logicielle spécifique. Si le programme est extrait de cette distribution et utilisé ou distribué selon les conditions de la licence du programme, toutes les parties auxquelles le programme est redistribué doivent bénéficier des droits accordés lorsque le programme est au sein de la distribution originale de logiciels.

B.1.9 Pas de contamination d'autres logiciels

La licence ne doit pas apposer de restrictions sur d'autres logiciels distribués avec le programme qu'elle protège. Par exemple, la licence ne doit pas exiger que tous les programmes distribués grâce au même medium soient des logiciels « open source ».

B.1.10 Exemples de licences

Les licences suivantes sont des exemples de licences que nous jugeons conformes à la définition de l'OpenSource : GNUGPL, BSD, XConsortium, et Artistic. C'est aussi le cas de la MPL.

Si vous avez des questions ou des suggestions veuillez les adresser en anglais au webmestre responsable de la version originale de ce document.

B.2 Justification de la définition de l'OpenSource

Le but de la définition de l'OpenSource est de protéger le processus de l'OpenSource, de s'assurer que le logiciel « open source » pourra être examiné par des pairs indépendants et suivre une évolution faite d'améliorations et de sélections continues, pour atteindre des niveaux de fiabilité et de puissance dont aucun éditeur de produit propriétaire ne peut se targuer.

Pour que ce processus d'évolution fonctionne, il nous faut contrer les motivations à court terme que certains pourraient avoir de cesser de contribuer au « patrimoine génétique du logiciel ». Cela signifie que les conditions de la licence doivent empêcher quiconque de fermer le logiciel et de n'autoriser que fort peu de gens à l'examiner ou à le modifier.

La définition de l'OpenSource n'est pas, et ne sera jamais, un hameçon pour pêcheur de droits d'utilisation. Tout un chacun est libre, et le demeurera, d'utiliser cette marque de certification s'il en remplit les conditions.

B.2.1 Libre redistribution

En contraignant la licence à imposer la libre redistribution, on ôte la tentation d'abandonner de nombreux gains à long terme pour gagner rapidement

de l'argent sur les ventes. Sans cette contrainte, les participants au mouvement subiraient de nombreuses pressions pour l'abandonner.

B.2.2 Code source

On impose l'accès à un code source non délibérément rendu peu clair car on ne peut alors pas faire évoluer un programme. Puisque notre but est de faciliter l'évolution des programmes, on impose que ces modifications soient facilitées.

B.2.3 Travaux dérivés

La seule possibilité de lire le code source ne suffit pas pour permettre un examen des pairs et une rapide évolution par sélection. Pour autoriser cette dernière, il faut donner l'autorisation d'expérimenter des modifications et de les distribuer.

B.2.4 Intégrité du code source de l'auteur

C'est une bonne chose que d'encourager de nombreuses améliorations, mais les utilisateurs ont le droit de savoir qui est à l'origine du logiciel qu'ils utilisent. Les auteurs et ceux qui font évoluer un logiciel ont aussi le droit, réciproque, de savoir ce qu'on leur demande de faire évoluer, et de protéger leur réputation.

C'est pourquoi une licence OpenSource doit garantir que le code source soit facilement accessible, mais peut demander qu'il soit distribué sous la forme de code source originel, accompagné des modifications à lui apporter. De cette manière, on peut mettre à disposition des modifications « non officielles » non amalgamées au code source de base.

B.2.5 Pas de discrimination sur l'identité de l'utilisateur

Pour tirer le meilleur profit du processus, il faut autoriser, à égalité, la plus grande diversité de personnes et de groupes à contribuer aux logiciels OpenSource. C'est pourquoi nous interdisons à toute licence OpenSource d'exclure qui que ce soit.

B.2.6 Pas de discrimination sur le domaine d'application

Le but premier de cette clause est d'interdire les pièges de licences qui empêchent d'exploiter commercialement des logiciels OpenSource. Nous voulons que les utilisateurs commerciaux rejoignent notre communauté, et ne s'en sentent pas exclus.

B.2.7 Distribution de la licence

Le but de cette clause est d'interdire la fermeture du logiciel par des moyens indirects tels que l'imposition d'un accord de non divulgation.

B.2.8 Pas de spécificité

Cette clause permet d'éviter un autre piège classique des licences.

B.2.9 Pas de contamination d'autres logiciels

Qui souhaite utiliser ou redistribuer des logiciels OpenSource doit avoir le droit d'appliquer à ses propres logiciels les conditions de son choix.

Annexe C

Note de Linus Torvalds à propos des appels système sous Linux

Linus Torvalds ajoute le préambule suivant à l'exemplaire de la GPL protégeant le code source de Linux et livré avec lui :

Notabene Ce copyleft (ou « gauche d'auteur ») ne concerne pas les programmes qui utilisent les services du noyau sous la forme d'appels système habituels utilisation du noyau considéré comme normale. Ils n'appartiennent donc pas à l'ensemble des « travaux dérivés ».

Vous remarquerez aussi que le copyright de la GPL appartient à la *Free Software Foundation*, mais que celui de l'instance du code qu'elle protège (le noyau Linux) m'appartient, ainsi qu'à ceux qui m'ont aidé à l'écrire.

Annexe D

Contributeurs

D.1 Brian Behlendorf

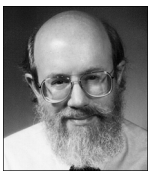
Brian Behlendorf ne ressemble en rien à l'idée qu'une personne non avertie se fait d'un hacker. Il est co-fondateur et membre principal de l'Apache Group. Apache est un logiciel libre pour serveur web (en Open Source) qui anime 53% des serveurs de l'Internet accessibles au public. Cela signifie que ce programme libre a conquis plus de parts de marché que les offres combinées de Microsoft, Netscape, et tous les autres éditeurs.

Brian participa durant quatre ans au développement d'Apache, aidant à guider la croissance du projet conjointement avec les autres membres de l'équipe d'Apache. Ce qui démarra comme une expérience intéressante est devenu un serveur web bien ficelé et totalement fonctionnel.

Il n'est pas le seul dans ce livre à s'être dédié à la musique, mais il est probablement le seul à avoir organisé des raves ou à avoir officié comme DJ dans des fêtes. Son site web recèle de merveilleuses ressources en matière de musique, de rave et de club. Il aime la lecture, et les livres ne traitant pas d'informatique récemment appréciés sont « Tao of Physics »de Capra et « Secrets, Lies and Democracy » de Chomsky.

Fin 1998, IBM a annoncé son support pour Apache sur sa toute dernière gamme d'AS/400, un réel tournant pour le projet Apache. Brian a commenté le ralliement d'IBM en disant qu'il était « content de ne pas avoir été le seul à penser que résidait là un commerce potentiel. Pas seulement un travail amusant, mais un modèle pour le commerce. Les gens s'y intéressent et constatent que le mode de développement de l'Open Source permet de meilleures réalisations informatiques. De plus la façon de faire est saine voire profitable ».

D.2 Scott Bradner



Scott Bradner a été impliqué dans la conception, la mise en œuvre et l'utilisation des réseaux de transmission de données de l'Université de Harvard depuis les premiers jours de l'ARPAnet. Il fut aussi présent dans la conception du High-Speed Data Network (HSDN) de Harvard, du Medical Area network (LMAnet) de Longwood et du NEARNET. Il a détenu, lorsqu'elle fût fondée, la chaire des comités techniques de LMAnet, NEARNET, et CoREN.

Scott est co-directeur du Transport Area de l'IETF, membre de l'IESG, et membre élu du conseil d'administration de l'Internet Society où il joue le rôle de vice-président aux Standards. Il fut aussi co-directeur du projet IETF IPng et est co-éditeur de « IPng : Internet Protocol Next Generation » paru chez Addison-Wesley.

Scott assure des services d'expertise technique auprès du bureau du prévôt de Harvard, où il livre ses conseils techniques et offre son expertise en matière de réseaux de données et de nouvelles techniques. Il gère aussi le Harvard Network Device Test Lab, est un orateur fréquent des conférences techniques ainsi qu'un journaliste pour l'hebdomadaire Network World, un formateur pour Interop, et il assure parfois des missions de conseil.

D.3 Jim Hamerly

Jim Hamerly est vice-président de la division produits grand public de Netscape Communications Corporation. En juin 1997 Netscape a acquis DigitalStyle Corporation, dont Jim était co-fondateur, président, et CEO.

Avant de fonder DigitalStyle, il fut vice-président en charge de l'ingénierie de Pages Software, Inc. où il coordonnait le développement de Pages, un outil bureautique et WebPages, le premier outil WYSIWYG pour la composition de pages web.

Jim a passé 15 ans chez Xerox occupé à diverses activités de RD et de développement produit, dont le plus récemment en tant que Deputy Chief Engineer de XSoft, une division logicielle de Xerox Corporation, où il était responsable de quatre lignes de produits.

Jim est titulaire d'un B.S., d'un M.S., et d'un Ph.D. en Ingénierie Électrique et Science Informatique du MIT, de l'UC Berkeley et de l'Université de Carnegie Mellon.

D.4 Kirk McKusick

Kirk McKusick écrit des livres et des articles. Il est aussi conseil et donne des cours sur des sujets liés à Unix et BSD. Durant son passage à l'Université de Californie à Berkeley, il a implémenté le Fast FS (système de fichiers rapide) 4.2BSD, et fut le Research Computer Scientist dans le groupe Berkeley Computer Systems Research (CSRG) chapeautant le développement et la diffusion de 4.3BSD et 4.4BSD. Ses domaines de prédilection sont le système de mémoire virtuelle et le système de fichiers. Il espère les voir fusionner un jour. Il a décroché son diplôme d'ingénieur en électricité à l'Université de Cornell, et a effectué son travail de diplôme à l'Université de Californie de Berkeley, où il obtint ses Masters en Informatique et Administration Commerciale, ainsi qu'un doctorat en Informatique. Il fut président d'Usenix Association, et est membre de l'ACM et de l'IEEE.

Durant son temps libre, il apprécie la natation, la plongée et collectionne les vins. Ses bouteilles se trouvent dans une cave à vin construite spécialement (accessible depuis le Web) dans les fondations de la maison qu'il partage avec Eric Allman, son colocataire depuis environ 19 ans.

D.5 Tim O'Reilly

Tim O'Reilly est le fondateur et CEO d'O'Reilly Associates, Inc., l'éditeur dont les ouvrages sont considérés comme des références en matière d'Open Source telles que Perl, Linux, Apache, et l'infrastructure de l'Internet. Tim a convoqué le premier « Sommet de l'Open Source » afin de rassembler les leaders de communautés Open Source majeures, et a été un promoteur actif du

mouvement Open Source en écrivant, discutant et donnant des conférences. Il est aussi membre du conseil d'administration de l'Internet Society.

D.6 Tom Paquin

Tom Paquin a d'abord rejoint IBM Research pour œuvrer sur un projet impliquant des processeurs parallèles, mais a fini par travailler sur un accélérateur graphique bitmap (dérivé de l'AMD 29116) pour ce qui était alors le nouveau PC. Après avoir bricolé un peu sur X6 et X9 au MIT et à l'Université Brown, il fut l'un de ceux qui permirent de livrer la première mouture commerciale de X11 avec l'Université de Carnegie Mellon.

Tom rejoignit Silicon Graphics, Inc. (SGI) en mai 1989, où la tâche ingrate d'intégrer GL et X lui fut confiée. Il entra chez Netscape aux côtés de Jim Clark et Marc Andreesson en avril 1994. Il y fut le tout premier responsable de l'ingénierie, pilotant son équipe dans les versions 1.0 et 2.0 de Mozilla. Désormais membre de Netscape, il travaille sur mozilla.org en tant que responsable de projet, arbitre des problèmes, et mystérieux leader politique.

D.7 Bruce Perens

Bruce Perens est un avocat de Linux et du logiciel open source depuis longtemps. Jusqu'en 1997 Bruce était à la tête du projet Debian, un effort bénévole pour créer une distribution de Linux ne comprenant que du logiciel open source.

Pendant son travail sur le projet Debian, Bruce a contribué à la maturation du projet social Debian (Debian Social Contract), un ensemble de conditions sous lesquelles un logiciel peut être considéré comme relevant d'une licence suffisamment libre pour être inclus dans la distribution Debian. Le contrat social Debian est l'ancêtre direct de l'actuelle définition de l'Open Source (Open Source Definition ou OSD).

Après s'être retiré de l'intendance de Debian, Bruce continua ses efforts d'évangélisation en faveur de l'Open Source en créant et dirigeant Software in the Public Interest, et en démarrant, avec Eric Raymond, l'Open Source Initiative.

Lorsqu'il n'évangélise pas activement pour le logiciel Open Source, Bruce travaille pour les studios d'animation Pixar.

D.8 Eric Steven Raymond

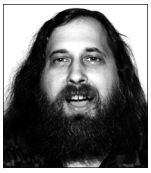
Eric Steven Raymond est un hacker de longue date. Depuis l'époque de l'ARPAnet vers la fin des années 1970, il a observé et pris part à l'émergence des cultures hacker et l'Internet avec émerveillement et fascination. Il a vécu sur trois continents et oublié deux langues avant l'âge de quatorze ans, et il aime à penser que cela renforça sa vision anthropologique du monde.

Il a étudié les mathématiques et la philosophie avant d'être séduit par les ordinateurs, et a aussi rencontré un certain succès en tant que musicien (en jouant de la flûte sur deux albums). Plusieurs de ses projets open source sont intégrés dans toutes les distributions majeures de Linux. Le plus connu est probablement fetchmail, mais il a aussi contribué de façon importante à GNU Emacs et ncurses. Il est actuellement le mainteneur de termcap, un de ces travaux apportant peu de reconnaissance mais qu'il est si important de bien faire. Eric est aussi ceinture noire de Tae Kwon Do et aime tirer au pistolet pour se détendre. Son arme favorite est le classique semi-automatique .45 modèle 1911.

Parmi ses travaux en tant qu'auteur, il a compilé le dictionaire intitulé « The New Hackers Dictionary » [1]) et est co-auteur de l'ouvrage « Introduction à GNU Emacs »(« Learning GNU Emacs ») paru chez O'Reilly. En 1997, il a posté un essai sur le Web intitulé « La cathédrale et le bazar » (« The Cathedral and the Bazaar ») qui est considéré comme un élément clé ayant conduit Netscape à ouvrir le code source de son navigateur.

Depuis lors Eric surfe adroitement sur la vague du logiciel Open Source. Récemment, il a disséqué l'histoire d'une série de mémos internes de chez Microsoft concernant Linux et la menace que constitue le logiciel open source selon Microsoft. Ces documents appelés Halloween (en raison de la date de leur découverte, le 31 octobre) furent à la fois une source d'amusement et la première confirmation d'une réaction qu'un grand conglomérat logiciel a montré face au phénomène Open Source.

D.9 Richard Stallman



Chaque auteur de texte publié dans ce livre a, d'une façon ou d'une autre, une dette envers Richard Stallman (RMS). Il a lancé voici 15 ans le projet GNU afin de protéger et renforcer le développement du logiciel libre. Un objectif établi du projet était de développer un système d'exploitation entier et des jeux complets d'utilitaires sous couvert d'une licence libre et ouverte de telle sorte à ce que personne n'ait plus à payer pour du logiciel.

En 1991, Stallman reçoit le prestigieux Grace Hopper Award de l'Association for Computing Machinery pour son développement de l'éditeur Emacs. En 1990 il a été récompensé par un MacArthur Foundation fellowship. Un doctorat à titre honorifique de la Royal Institute of Technology de Suède lui a aussi été décerné en 1996. En 1998 il a partagé avec Linus Torvalds le Pioneer award de l'Electronic Frontier Foundation.

Il est désormais davantage connu pour son évangélisme en faveur du logiciel libre que pour le code auquel il a contribué.

Comme toute personne totalement dévouée à sa cause, Stallman a suscité la controverse dans la communauté dont il fait partie. Son insistance à répéter que les termes *Open Source software* ont été choisis spécifiquement pour occulter les aspects liés à la liberté que donne un logiciel libre est une parmi les nombreuses positions qu'il a adoptées et qui ont conduit certains à l'étiqueter comme un extrémiste. Il assume tout cela sans faire le moindre effort, ainsi que

1. aussi connu sous le nom de « Jargon File »

peut en témoigner quiconque l'a rencontré, revêtu de l'habit de son alter ego, saint iGNUcius de l'Église d'Emacs.

Beaucoup ont dit, « Si Richard n'avait pas existé, il aurait fallu l'inventer ». Cet éloge est une reconnaissance honnête du fait que le mouvement Open Source n'aurait pu émerger sans le mouvement du logiciel libre que Richard a popularisé et en faveur duquel il milite aujourd'hui encore.

En plus de ses positions politiques, Richard est connu pour sa participation à bon nombre de projets logiciels. Les deux plus proéminents sont le compilateur GNU C (GCC) et l'éditeur Emacs. GCC est de loin le plus porté et le plus populaire des compilateurs sur la planète. Cependant, RMS est avant tout connu pour son éditeur Emacs. Dire de l'éditeur Emacs qu'il s'agit d'un éditeur est comme dire de la Terre qu'elle est un joli tas de terre. Emacs est un éditeur, un navigateur web, un lecteur de nouvelles Usenet, un agent de courrier électronique, un gestionnaire d'information personnelle, un programme de composition, un éditeur pour la programmation, un éditeur hexadécimal, un processeur de textes, et un certain nombre de jeux vidéo. Beaucoup de programmeurs utilisent un évier de cuisine comme icône de leur copie d'Emacs. D'autres démarrent Emacs et ne le quittent plus, Emacs permettant de faire tout ce que leur ordinateur sait faire d'autre. Emacs, vous le découvrirez, n'est pas seulement un programme. C'est une religion. Et RMS est son saint.

D.10 Michael Tiemann

Michael Tiemann est un fondateur de Cygnus Solutions™. Michael a commencé à faire des contributions à la communauté de développement de logiciel à travers son travail sur le compilateur GNU C (qu'il a porté sur SPARC et différentes autres architectures RISC), le compilateur GNU C++ (dont il est l'auteur), et le débogueur GDB (qu'il a amélioré afin qu'il supporte le langage de programmation C++ et qu'il a porté sur SPARC). Incapable de convaincre une quelconque société de fournir un service commercial d'assistance pour ce nouveau logiciel open source, il a co-fondé Cygnus Solutions™ en 1989. Aujourd'hui, Michael est un orateur et un expert reconnu dans le domaine des modèles de logiciel open source et du leur négoce, et il continue à chercher des solutions techniques et commerciales qui rendront les dix années à venir aussi excitantes et gratifiantes que les dix années écoulées.

Michael est titulaire d'un B.S. en CSE obtenu en 1986 de la Moore School of Engineering, à l'Université de Pennsylvanie. Entre 1986 et 1988, il a travaillé chez MCC à Austin, Texas. En 1988, il est entré à la Stanford Graduate School (EE) et est devenu candidat à un Ph.D. au printemps 1989. Michael a abandonné son programme de Ph.D. fin 1989 pour démarrer Cygnus™.

D.11 Linus Torvalds

Il a créé Linux, bien sûr. C'est comme de dire « Engelbart a inventé la souris ». Il est certain que les implications à long terme de cet article Usenet :

From : torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups : comp.os.minix
Subject : Gcc-1.40 and a posix-question
Message-ID : <1991Jul3.100050.9886@klaava.Helsinki.FI>
Date : 3 Jul 91 10 :00 :50 GMT

Salut !
Pour un projet sur lequel je travaille (dans minix), je m'intéresse au normes POSIX. Qui peut m'indiquer où trouver (de préférence sous forme de fichier) la plus récente version ? Une adresse de site FTP serait très bien.

ne lui apparurent pas clairement lorsqu'il le publia.

Linus ne pouvait en effet pas prévoir que son projet irait d'un petit passe-temps vers un système comptant 7 à 10 millions d'adhérents, concurrent majeur du produit de la plus importante société d'édition de logiciel.

Depuis l'adoption massive de Linux et sa croissance tel un feu de forêt à travers l'Internet - ainsi 26% des serveurs de l'Internet fonctionnent sous Linux (le concurrent le plus proche étant Microsoft avec 23%) - la vie de Linus Torvalds a changé. Il a quitté sa Finlande natale pour la Silicon Valley, où il travaille pour Transmeta Corporation. À propos de son travail chez Transmeta, il dit seulement que cela n'implique pas Linux et que c'est « vachement cool ».

Il a deux enfants et un brevet (portant sur un « contrôleur mémoire de microprocesseur pour la détection d'une erreur de spéculation sur la nature physique d'un composant adressé »), et a été invité au plus prestigieux événement finlandais, le bal de l'Indépendance du président.

Sa personnalité l'empêche de s'accaparer les honneurs de ce qui ne vient pas de lui, et Linus est prompt à mettre en avant que sans l'aide de bien d'autres, Linux n'aurait pu devenir ce qu'il est aujourd'hui. Des programmeurs talentueux comme David Miller, Alan Cox, et d'autres ont tous été les instruments du succès de Linux. Sans leur aide et celle d'un nombre incalculable d'autres, le système d'exploitation Linux n'aurait pas atteint les sommets qu'il occupe aujourd'hui.

D.12 Paul Vixie

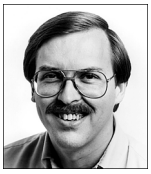
Paul Vixie est à la tête de Vixie Enterprises. Il est aussi le président et fondateur de l'Internet Software Consortium, le berceau de bind, inn, et du serveur dhcpd. Paul est l'architecte principal de bind, le serveur de noms le plus populaire. Inn est le paquetage serveur de news de l'Internet et dhcp permet la configuration dynamique de divers paramètres réseau.

Il est l'auteur de Vixie cron, le démon cron par défaut de Linux, et de la plupart du reste du monde. Cela signifie qu'il est probablement responsable de l'étrange bruit que fait votre ordinateur à 1 h du matin chaque nuit.

Paul est l'auteur de l'ouvrage « Sendmail : Theory and Practice ». Sa société gère un réseau pour Commercial Internet Exchange, et dirige la lutte contre le

spam avec MAPS (Mail Abuse Protection System). MAPS est constitué d'une liste « trou-noir » en temps réel (où les spammeurs voient leurs adresses e-mail disparaître dans le saint broyeur de bits) et des propositions pour la sécurité du transport du courrier électronique.

D.13 Larry Wall



Larry Wall est l'auteur de quelques programmes open source parmi les plus populaires pour Unix, incluant le lecteur de news rn, le programme polyfonctionnel patch et le langage de programmation Perl. Il est aussi connu pour metaconfig, un programme qui écrit des scripts Configure, et pour le jeu d'arcade warp, dont la première version a été écrite en BASIC/PLUS à l'Université de Seattle Pacific. Par sa formation, Larry est en fait un linguiste, ayant fait un passage à la fois à U.C. Berkeley et à UCLA en tant qu'étudiant diplômé (assez curieusement, alors qu'il était à Berkeley, il n'a pas du tout été impliqué dans le développement d'Unix qui s'y opérait).

Larry a été un programmeur pour le JPL[2]. Il a aussi passé du temps chez Unisys, en jouant avec tout depuis les simulateurs d'événements discrets jusqu'aux méthodes de développement. Perl naquit là-bas, alors qu'il tentait de mettre en place un système d'administration de sites très éloignés via une ligne chiffrée à 1200 bit/s en utilisant une version modifiée de Netnews.

Actuellement Larry a mis ses talents au service d'O'Reilly, qu'il conseille dans les domaines liés à Perl.

D.14 Bob Young

Bob Young a toujours été quelque part entre une énigme et une légende dans la communauté de l'Open Source. Il est homme d'affaires et non hacker, et a été décrit pendant longtemps dans les cercles Linux comme l'adulte mythique qui organisa les efforts de l'équipe de hackers employée par Red Hat.

Bob a passé les vingt premières années de sa vie professionnelle dans le domaine de la location d'ordinateurs, dirigeant deux entreprises différentes avant d'entrer dans le monde de Linux. Il fut l'éditeur original de Linux Journal avant que Phil Hughes et SSC en prennent la charge. Bob a rejoint Red Hat en faisant la promesse que les associés d'alors, dirigés par Marc Ewing, n'auraient pas à se soucier de la gestion des aspects financiers au sein de la société. Il a appliqué les règles de promotion plus habituelles chez Gap ou Harley-Davidson au monde du logiciel libre. Ceci était parfaitement adapté à une société dont l'activité principale est d'emballer un produit : le logiciel Open Source.

Red Hat voulait à l'origine fournir des versions OEM de Linux dont elle souhaitait approvisionner les sociétés éditant des systèmes commerciaux, plutôt que de se livrer à du marketing ou de la vente directe de ses propres produits. Ses partenaires commerciaux ayant échoué à mettre leurs produits sur le marché

2. Jet Propulsion Laboratory

dans les temps, Red Hat s'est mise à vendre sa propre distribution, afin que ses employés (c'est ainsi que va l'histoire) soient assurés d'un salaire pour vivre.

Red Hat a récemment reçu un soutien financier par l'investissement de sociétés de capitaux risque, ainsi que de Netscape et Intel. Ironie de l'histoire, c'est la vente directe de ses produits qui a fait le succès de RedHat, alors que tel n'était pas le but à l'origine.

D.15 Chris DiBona

Chris DiBona utilise Linux depuis 1995. Il est très actif dans la communauté Linux. Il fut volontaire pour être le webmestre de Linux International et est aussi le coordinateur des fonds et subventions pour le développement de Linux International. Il est fier de travailler en tant que directeur du Marketing Linux pour VA Research Linux systems et est aussi vice-président du Silicon Valley Linux Users Group le plus grand au monde.

En plus de ses activités Linux, ses écrits et revues de livres ont pu être appréciées dans The Vienna Times, Linux Journal, Tech Week, Boot Magazine (désormais Maximum PC), et nombre d'autres publications en ligne. Qui plus est, il fut éditeur durant deux ans du Terrorist Profile Weekly, un hebdomadaire géopolitique distribué à 20000 lecteurs.

Son site web personnel se trouve à http://www.dibona.com.

D.16 Sam Ockman

Sam Ockman est le président de Penguin Computing, une société spécialisée dans les systèmes Linux sur mesure. Il est le président de LINC, l'exposition et conférence Linux internationale qui a fusionné avec LinuxWorld. Sam est un expert en installation et configuration de systèmes Linux, ainsi qu'en Perl, qu'il a enseigné à l'Université de Californie, Berkeley Extension School. Il coordonne aussi les orateurs pour le Silicon Valley Linux User Group et le Bay Area Linux User Group. Sam a édité des ouvrages sur Unix et Perl et écrit chaque mois un article sur Linux. Il est diplômé de Stanford en Ingénierie des Systèmes Informatiques et Science Politique. Sam est très fier d'avoir remporté le prix Ram's Head Dorthea Award en tant que meilleur acteur dramatique, lors de son passage à Stanford.

D.17 Mark Stone

Mark Stone utilise Linux comme son système d'exploitation principal depuis la version 1.0.8 du noyau. Il écrivit son premier programme d'ampleur vers la fin des années 70 : un compilateur Algol pour le PDP-11/70. Aujourd'hui il préfère les scripts aux langages compilés ; son langage favori est Tcl.

Actuellement Mark est responsable de l'édition Open Source pour O'Reilly. Avant de rejoindre le monde de la publication il fut professeur de philosophie et détient un Ph.D. de l'Université de Rochester. Durant son séjour à l'université il a étudié la théorie du chaos et la philosophie des sciences. Sur bien des aspects, son travail n'a donc pas vraiment changé.