N°163 SEPTEMBRE 2013

GNU LINUX MAGAZINE / FRANCE

NOUVEAUTÉS / KERNEL 3.8

Découvrez la notion d'espaces de noms du noyau et la nouvelle API permettant de créer de nouveaux espaces UID/GID p.10

ADMINISTRATION ET DÉVELOPPEMENT SUR SYSTÈMES OPEN SOURCE ET EMBARQUÉS

GESTION / LOGS

Utilisez le framework Logstash pour automatiser votre gestion et vos analyses de logs p.18

C / ANALYSE

Analysez le fonctionnement d'un simulateur de vol complet tenant en une poignée de lignes de C p.70

PYTHON / DEBUG

Espionnez, observez et tracez l'exécution d'un script Python à l'aide du module Smiley p.79

DEBIAN / WHEEZY

Retour sur les nouveautés inédites de la dernière version majeure de Debian GNU/Linux p.04

L 19275 - 163 - F: 7,50 € - RD

HA / HAUTE DISPO

Mettez en œuvre IPVS et KeepAlived, des solutions de haute disponibilité simples et efficaces pour vos serveurs p.28

SGBD / SÉCURITÉ

Étudiez le moteur de base de données H2 puis ajoutez le contrôle d'accès au niveau ligne ainsi que l'accès distant p.60

BOOT / RÉSEAU

Postes de travail, machines de test, serveurs…
Passez-vous de disque dur, de CD et de clé USB et…

CRÉEZ VOS POSTES SANS DISQUE p.38

1. Le boot réseau/diskless PXE, Etherboot et romboot
2. iPXE : Mise en œuvre et installation d'un code de boot sur carte réseau
3. Configuration du système de fichiers racine en NFS
4. Utilisation des fonctionnalités avancées d'iPXE

SMARTPHONES / GUI

Ergonomie des interfaces utilisateur : Android, Windows Phone et iOS au banc d'essai (suite et fin) p.46

JAVA / POO

Factory, interfaces, aspects, singletons... Java, héritier du C/C++ et de Smalltalk, est-il vraiment un langage objet ? p.35

France METRO : 7,50 € – CH : 13 CHF – BEL/PORT.CONT : 8,50 € – DOM : 8 € – CAN : 13,75 $ cad – NCAL/S : 1000 CFP – POL/S : 1100 CFP – POL/A : 1500 CFP – TUNISIE : 16,50 TND – MAR : 95 MAD

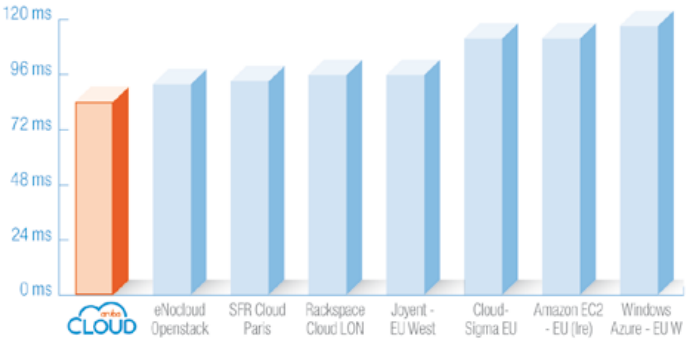
Aruba Cloud, la meilleure offre de Cloud Computing du marché

Le plus performant et le moins cher des Clouds Publics

Fournisseur	Coût mensuel estimé	Performances	Sécurité	Flexibilité
Meilleure proposition	aruba CLOUD	aruba CLOUD	Rackspace	Amazon Web Services
aruba CLOUD	69 €	★★★★½	★★★☆☆	★★★½☆
Windows Azure	87 €	★★★★☆	★★★½☆	★★★★☆
Amazon Web Services	96 €	★★★½☆	★★★★☆	★★★★½
CloudSigma	135 €	★★★★☆	★★★½☆	★★★☆☆
Rackspace	157 €	★★★★☆	★★★★☆	★★★★☆

Source www.cloudscreener.com
sur les paramètres standards d'évaluation des offres de Cloud Computing.

Classé n°1 en temps de réponse, temps de connexion et disponibilité



Source : Analyse Cedexis réalisée du 5 mai au 5 juin 2013 depuis la France.
www.cedexis.com

TESTEZ GRATUITEMENT NOTRE OFFRE

Aruba, le bon partenaire pour bénéficier de la puissance d'un acteur majeur qui considère que chaque client, dans chaque pays, est unique. **MY COUNTRY. MY CLOUD.**

aruba CLOUD
arubacloud.fr | TÉL : 0810 710 300
(COÛT D'UN APPEL LOCAL)

SOMMAIRE N°163

GNU/Linux Magazine France
est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND

B.P. 20142 – 67603 Sélestat Cedex
Tél. : 03 67 10 00 20 – **Fax** : 03 67 10 00 21
E-mail : lecteurs@gnulinuxmag.com
Service commercial : abo@gnulinuxmag.com
Sites : www.gnulinuxmag.com – boutique.ed-diamond.com

Directeur de publication : Arnaud Metzler
Rédacteur en chef : Denis Bodor
Secrétaire de rédaction : Véronique Sittler
Réalisation graphique : Jérémy Gall

Responsable publicité : Valérie Fréchard, Tél. : 03 67 10 00 27
v.frechard@ed-diamond.com
Service abonnement : Tél. : 03 67 10 00 20
Impression : pva, Druck und Medien-Dienstleistungen GmbH, Landau, Allemagne
Distribution France : (uniquement pour les dépositaires de presse)
MLP Réassort : Plate-forme de Saint-Barthélemy-d'Anjou. Tél. : 02 41 27 53 12
Plate-forme de Saint-Quentin-Fallavier. Tél. : 04 74 82 63 04
Service des ventes : Distri-médias : Tél. : 05 34 52 34 01
IMPRIMÉ en Allemagne - PRINTED in Germany
Dépôt légal : À parution, N° **ISSN** : 1291-78 34
Commission paritaire : K78 976

Périodicité : Mensuel
Prix de vente : 7,50 €

FSC www.fsc.org MIXTE Papier issu de sources responsables FSC® C015136

Membre April www.april.org

ÉDITORIAL

Ah les vacances... Le soleil, les projets personnels, les balades en nature, la découverte de nouveaux horizons, les siestes (pas trop longues tout de même)... Eh bien, tout ceci est terminé ! Nous voici en septembre et il est temps de faire preuve de motivation et de raviver la flamme du sysadmin et du développeur qui finalement ne vous aura pas quitté durant cette période estivale.

Pour bien entamer la rentrée, le GLMF de ce mois est aussi diversifié que dense (finalement, la rédaction n'a pas fait beaucoup de ballades ou de siestes cet été). Gestion des logs, programmation et obfuscation C, débogage Python, GUI mobile, boot réseau, flashage de bootrom, haute disponibilité, extension des fonctionnalités d'un SGBD ou encore nouvelle API du noyau Linux 3.8, voici autant de sujets qui devraient vous occuper et vous empêcher de dormir dans les prochaines semaines.

En vérité, je vous le dis, les nouveaux horizons à explorer sont là où on les cherche, là où on repousse les limites et là où on canalise son énergie. Il ne tient qu'à vous de partir à leur découverte et le magazine que vous tenez entre vos mains, sans fausse modestie aucune, est, je pense, un excellent point de départ pour cette rentrée 2013.

Je profite du peu d'espace à ma disposition ici pour glisser subtilement (ou pas) une petite information pratique : notre boutique en ligne a subi récemment une violente refonte/évolution en passant d'une vieillissante plateforme à quelque chose de bien plus riche en fonctionnalités. D'autres évolutions (comme la gestion complète en ligne de vos abonnements) sont sur le point d'être ajoutées, mais je vous invite dès aujourd'hui à aller y faire un tour, ne serait-ce que pour constater l'étendu des changements :)

Sur ces belles paroles, je vous souhaite une vivifiante et amusante rentrée et... pardon pour vos nuits.

Denis Bodor

DEBIAN WHEEZY 7.0

par Carl Chenet
[Architecte système GNU/Linux freelance, contributeur Debian et Pythoniste]

La nouvelle version stable de la distribution à la spirale, numérotée 7.0 et nommée Wheezy, du nom du manchot en caoutchouc avec une cravate rouge du film d'animation Toy Story, a été publiée le 4 mai 2013, après 26 mois de développement.

L'arrivée de cette nouvelle version stable a été un événement majeur pour la communauté du logiciel libre. Cette distribution équipera en effet pour de nombreuses années à venir un parc important de serveurs et de postes de travail dans le monde. Mais au-delà de ses utilisateurs directs, Wheezy servira également de base à de très nombreuses distributions dérivées.

Véritable socle sur lequel repose une imposante partie de l'écosystème du logiciel libre, cette version de Debian offre de nombreux nouveaux programmes et des fonctionnalités inédites.

1 Quelques généralités à propos de Debian

Le projet Debian fêtera son vingtième anniversaire cette année. Initié par Ian Murdock, alors étudiant, qui fusionna son prénom et celui de sa compagne d'alors (Debra) pour créer le nom « Debian », le projet a publié sa première version majeure 1.1 « Buzz » le 17 juin 1996. Cette distribution GNU/Linux – et depuis Squeeze également GNU/kFreeBSD (voir plus bas) – est donc l'une des plus anciennes et des plus pérennes.

Historiquement très lié au mouvement GNU de Richard Stallman qui le soutint à ses débuts, le projet Debian en retira une conscience forte du respect du logiciel libre, incarné par un texte définissant les principes du logiciel libre selon Debian, qui sont au centre de son contrat social, agréé par l'ensemble des développeurs Debian.

2 Transition entre les deux versions stables de Debian

Squeeze 6.0 est devenue la vieille stable (*old stable*) le jour de la sortie de Wheezy. Après sa sortie officielle le 6 février 2011 et 7 publications mineures (*point releases*), elle sera encore maintenue un an pour permettre une migration en douceur de ses utilisateurs vers la nouvelle version stable. Au-delà, les dépôts de paquets et images ISO de Squeeze resteront accessibles via le site des anciennes archives de Debian **[1]**. Plus aucun support ne sera par contre assuré. Une documentation complète sur la migration de Squeeze vers Wheezy accompagne les notes de publication de Wheezy **[2]**.

3 Le socle de Wheezy

Si la distribution stable de Debian est le socle d'une partie importante du mouvement du logiciel libre depuis de nombreuses années, c'est tout d'abord grâce au large support et aux fonctionnalités impressionnantes qu'elle offre dès l'installation. Le projet Debian a en effet vocation à produire le système d'exploitation universel, couvrant le spectre de besoins le plus large possible. Qu'il s'agisse des architectures supportées, de la gestion dès l'installation des différents matériels et configurations logicielles, Wheezy fait preuve d'une diversité et d'une solidité qui combleront les attentes de ses utilisateurs.

3.1 Architectures supportées

Debian Wheezy est disponible pour un grand nombre d'architectures matérielles et logicielles :

- PC 32-bits, identifiée par la dénomination **`i386`** ;
- PC 64-bits, identifiée par la dénomination **`amd64`** ;
- SPARC, identifiée par la dénomination **`sparc`** ;
- PowerPC, identifiée par la dénomination **`powerpc`** ;
- MIPS, appellation qui recouvre en fait deux architectures, à savoir *big-endian* (identifiée par la dénomination **`mips`**) et *little-endian* (identifiée par la dénomination **`mipsel`**) ;
- Intel Itanium, identifiée par la dénomination **`ia64`** ;
- S/390, identifiée par la dénomination **`s390`** ;
- ARM EABI, identifiée par la dénomination **`armel`** ;
- IBM System z, identifiée par la dénomination **`s390x`** ;
- ARMv7, identifiée par la dénomination **`armhf`** ;

Si vous utilisez un PC acheté récemment, vous vous orienterez par défaut vers l'architecture PC 64-bits, identifiée par la dénomination **`amd64`**.

Deux architectures logicielles font leur apparition dans Wheezy. L'architecture pour les machines IBM System z version 64 bits a pour vocation de remplacer la s390 pré-existante. L'architecture ARMv7 vient quant à elle compenser les manques de l'ancien port ARM EABI, quant au support des unités de calcul en virgule flottante embarquées sur les cartes et périphériques ARM, mais aussi les nouvelles fonctionnalités récentes des processeurs ARM.

Introduite dans Squeeze comme démonstration technologique, Debian GNU/kFreeBSD semble s'installer durablement parmi les architectures supportées, toujours en deux versions, à savoir PC 64-bits, identifiée par la dénomination **kfreebsd-amd64** et PC 32-bits, identifiée par la dénomination **kfreebsd-i386**.

Qualifiées de démonstrations technologiques, certaines fonctionnalités sont encore manquantes par rapport aux architectures matérielles équivalentes en GNU/Linux, mais la grande majorité des fonctionnalités et de la suite logicielle Debian sont disponibles sur ces plateformes.

3.2 Support d'architectures multiples

Debian 7.0 propose la prise en charge multiarchitecture, permettant à différentes bibliothèques et binaires prévus pour différentes architectures de s'exécuter sur un même système. Le cas le plus courant d'utilisation est l'utilisation conjointe d'éléments provenant des architectures 32 bits et 64 bits.

3.3 Amorçage par dépendances

L'ordonnancement des scripts de démarrage reposant sur un système de dépendances, introduit dans Squeeze, est maintenant toujours actif par défaut avec Wheezy. Cette fonctionnalité se base sur l'entête de vos scripts de démarrage au format *Linux Standard Base* (LSB). Pour en faire profiter vos scripts « maison » et les intégrer correctement à ce nouveau système, veillez à bien respecter ce prérequis.

3.4 Prise en charge de Systemd

Systemd, le système d'initialisation doté de fonctionnalités avancées de surveillance, d'enregistrement et d'initiation des services, est pris en charge par Debian 7.0. Encore considéré comme une démonstration technologique, plus de 50 paquets fournissent néanmoins une prise en charge native de Systemd. Le paquet **systemd** peut être installé conjointement avec le paquet **sysvinit** et activé avec l'option du noyau **init=/bin/systemd**.

3.5 Multimédia

La prise en charge du multimédia fait un bond en avant avec Wheezy, ce qui devrait avoir pour effet de réduire le recours aux dépôts de paquets tiers dans ce domaine. FFmpeg est remplacé par **libav**, jugé plus conservateur dans son modèle de développement et donc, plus en phase avec le développement de Debian, s'interfaçant avec **mencoder**, **mplayer**, **vlc**, **transcode**. **lame** se charge du support des codecs supplémentaires au niveau de l'encodage MP3, **xvidcore** pour l'encodage vidéo MPEG-4 ASP, **x264** pour l'encodage vidéo au format H.264/MPEG-4 AVC, **vo-aacenc** pour l'encodage audio au format AAC.

3.6 Sécurité renforcée (hardening)

Plusieurs paquets ont été construits en activant les attributs de renforcement de **gcc**, ceci afin de renforcer la sécurité face à des problèmes comme la protection de pile ou les emplacements prévisibles de valeurs en mémoire, avec un important effort sur les paquets de l'installation de base, les démons accessibles par le réseau - et donc exposés aux attaques - ainsi que sur les paquets ayant été compromis dans le passé.

3.7 Système de contrôle d'accès imposé AppArmor

Debian Wheezy prend en charge AppArmor, dont le but est de contrôler les accès des programmes à des fichiers grâce à un système de règles, dans l'esprit de SELinux embarqué sur les systèmes Red Hat et ce, afin d'améliorer la sécurité d'un système de manière préventive. AppArmor est désactivé par défaut sous Debian Wheezy.

3.8 Composants de base et Debian-Installer

Nous présentons ici différents programmes et leur version respective présents dans l'installation de base de Wheezy :

- Noyau Linux 3.2 et noyau FreeBSD 8.3,
- Glibc 2.13,
- Serveur de messagerie électronique par défaut Exim 4.80.

3.8.1 Changements du Debian-Installer

Le Debian-Installer (D-I) est le programme permettant la bonne installation de Debian sur votre ordinateur. Ce dernier est disponible en trois modes : texte, GTK2 et à partir de Wheezy la synthèse vocale, pour les personnes malvoyantes n'utilisant pas d'écran Braille. Une douzaine de langues sont supportées dans ce dernier mode.

Très flexible, le D-I supporte une grande variété de matériels et vous permet d'adapter différents paramètres selon vos besoins ou envies.

D-I est aussi un énorme effort de traduction, ce dernier assurant le support de pas moins de 74 langues (soit 3 de plus que pour Squeeze).

Parmi les fonctionnalités et modifications les plus importantes à retenir :

- Le support des architectures **armhf** et **s390x** par l'installeur ;
- Le bon fonctionnement de l'installeur via le réseau sur des architectures uniquement IPv6 ;
- Possibilité d'installer à travers un réseau sans fil chiffré en WPA ;
- Ext4 est le système de fichiers par défaut ;
- Support du Btrfs en tant que démonstration technologique,
- Installation possible en mode UEFI et non plus forcément en BIOS émulé.

3.9 Changements dans les systèmes de fichiers temporaires

Les systèmes de fichiers temporaires sous Squeeze étaient peu organisés au niveau de leurs points de montage. Wheezy rationalise cette situation en rendant systématique leur montage sous le répertoire **/run** à l'exception de **/tmp**. Le système de fichiers temporaire monté dans **/lib/init/rw** pour Squeeze est pour sa part supprimé dans Wheezy. Les nouveaux points de montage sont **/run**, **/run/lock** et **/run/shm**. Il est à noter que lors de la migration de Squeeze vers Wheezy, votre configuration sera automatiquement modifiée.

3.10 GNOME 3

GNOME 3 est pour Debian Wheezy le bureau utilisateur par défaut. Modification importante de l'interface, l'ergonomie est également profondément remaniée. Il s'agit du changement au niveau interface graphique qui surprendra sans doute le plus les utilisateurs. Bien que les changements soient très importants, il est possible de revenir à un thème plus habituel pour les utilisateurs GNOME de longue date via l'option ***Gnome Classic*** que l'on peut sélectionner au moment du choix de l'utilisateur offert par gdm3.

3.11 Offre autour du Cloud

Pour coller aux besoins actuels de ses utilisateurs en matière d'offre Cloud, Debian Wheezy offre les suites OpenStack et Xen Cloud Platform.

3.12 Debian Live

Comme ce fut le cas pour Squeeze, le projet Debian propose Wheezy sous la forme d'un live CD permettant de tester la distribution sans l'installer sur votre système. Deux architectures sont supportées : i386 et amd64. Plusieurs déclinaisons sont disponibles, à savoir standard ou secours (pour dépanner une installation qui ne démarre plus correctement), mais aussi celle proposant votre bureau préféré par défaut, comme GNOME, KDE, LXDE et Xfce.

Il est à noter qu'il est possible d'utiliser ce live CD à partir d'un classique cédérom ou DVD, mais aussi d'une clé USB, d'un disque dur ou via le réseau.

On trouve une information abondante sur la page de Debian Live **[3]**. Cette page offre toutes les informations nécessaires pour obtenir les images ISO du Debian Live pour un nombre important de médias, mais aussi toute la documentation nécessaire et exhaustive afin de vous aider à créer vos propres live CD.

4 Logiciels empaquetés

Les deux années consacrées au développement de cette nouvelle version par les membres du projet Debian ont été bien employées. Wheezy est riche en nouveaux programmes. Preuve de ce dynamisme, le nombre de logiciels empaquetés a considérablement augmenté. Ce dernier était de 29050 pour Squeeze, il est de 37493 pour Wheezy, sachant que plus de 4125 des paquets présents dans Squeeze ont été supprimés pour diverses raisons. Le tout tient sur 61 à 69 cédéroms (selon les architectures), ou entre 9 et 10 DVD dont les images ISO sont disponibles au téléchargement. Cet énorme choix assure aux utilisateurs de trouver dans Wheezy leur logiciel préféré en version stable, qui peut être facilement installé via le gestionnaire de paquets, avec le support associé par le mainteneur Debian chargé du paquet, qui est à votre écoute via le système de rapport de bugs. Ce support et ce suivi offerts par les mainteneurs aux utilisateurs est un avantage trop souvent négligé à mettre au crédit du projet Debian.

Rappelons au sujet de la gestion des paquets que **aptitude** est désormais le gestionnaire interactif de paquets en ligne de commandes à préférer pour votre utilisation quotidienne. En non-interactif, **apt-get** reste la commande de choix, en particulier pour les mises à jour entre deux versions majeures.

4.1 Côté serveur

Les principaux serveurs qui tournaient sur Squeeze ont été mis à jour. Wheezy offre de nouvelles versions comportant de nombreuses évolutions majeures. Parmi les paquets les plus utilisés : Serveur web Apache 2.2.22, Serveur DNS BIND 9.8.1, Serveur web Lighttpd 1.4.31, Server web Nginx 1.2.1, OpenSSH 6.0p1, Samba 3.6, Annuaire OpenLDAP 2.4.31, Serveur de messagerie électronique Postfix 2.9, Tomcat 6.0.35.

4.2 Côté base de données

Les bases de données ne sont pas en reste avec Wheezy, avec principalement MySQL 5.5, mais aussi PostgreSQL 9.1 et SQLite 3.7.13. Ces différentes versions ont largement fait leurs preuves tout en offrant de nombreuses nouvelles fonctionnalités par rapport aux versions présentes dans Squeeze.

4.3 Côté poste de travail

Wheezy propose d'importantes évolutions au niveau de votre poste de travail, la plus importante étant l'intégration de GNOME 3 dont nous avons déjà parlé. Debian présente un nouveau thème graphique « Joy » réalisé par Adrien Aubourg à la suite d'un concours. Le serveur graphique est X.Org en version 7.7. Présentons rapidement les principales mises à jour.

4.3.1 Environnements de bureau

L'environnement de bureau par défaut est GNOME, en version 3.4. Parmi les autres disponibles nous trouvons KDE 4.8.4 et Xfce 4.8.

4.3.2 Éditeurs de texte

Debian ne serait pas ce qu'il est sans eux : Emacs23, Vim 7.3, également disponible Nano en version 2.2.6.

4.3.3 Navigateurs web

Debian 7.0 propose les navigateurs web suivants : Iceweasel (Firefox démarqué) 10.0.12, Chromium 26.0 (version libre du navigateur web Chrome de Google), Lynx 2.8.8.

Il est à noter que le navigateur Galeon a été supprimé de Wheezy, ce dernier n'étant plus maintenu par ses développeurs officiels.

4.3.4 Clients de messagerie

Parmi les nouveautés les plus importantes des clients de messagerie disponibles dans Wheezy, nous trouvons Icedove (Thunderbird démarqué) en version 10.0.12. Evolution sera quant à lui proposé en version 3.4.4. Les *aficionados* de la ligne de commandes pourront utiliser Mutt 1.5.21.

4.3.5 Suites et outils bureautiques

Squeeze propose LibreOffice en version 3.5.4 et KOffice 2.4.3. Parmi les outils bureautiques également disponibles : GNUcash 2.4, GNUmeric 1.10, Abiword 2.9.

4.3.6 Messagerie instantanée

De nombreux clients de messagerie instantanée sont à la disposition des utilisateurs de Squeeze. Parmi eux : Pidgin 2.10, Gajim 0.15.1, Kmess 2.0.6, Xchat 2.8.8, Irssi 0.8.15, Minbif 1.0.5.

Qu'il soit multi-protocole ou spécialisé, il y a de fortes chances que votre client de messagerie instantanée présente une version récente dans la nouvelle Debian.

4.4 Côté programmation

Wheezy apporte aux programmeurs le nécessaire pour faire de leur station de travail une machine de développement efficace. Parmi les compilateurs et interpréteurs disponibles : GCC 4.7.2, Python 2.7 par défaut, également disponible Python 3.2, Perl 5.14, Ruby 1.9.3, PHP 5.4.4, OpenJDK 7, Eclipse 3.8.0.

4.5 Côté multimédia

Wheezy sera très apprécié des utilisateurs et producteurs de contenus multimédias. De nombreux lecteurs et outils de manipulation de contenus multimédias sont apparus ou ont été mis à jour. Parmi eux : FFmpeg 0.8.6, mplayer 1.0rc4, VLC 2.0.3, Guayadeque 0.3.5, gmusicbrowser 1.1.9, Ardour 2.8.14, Audacity 2.0.1.

Si vous souhaitez vérifier la disponibilité de votre logiciel multimédia préféré dans Wheezy, je vous encourage à consulter la page suivante **[4]**, qui fournit la liste complète des programmes multimédias dans Debian.

Que ce soit pour un serveur, en bureautique, dans le cadre d'une machine de développement ou d'une station multimédia, Wheezy offre à ses utilisateurs un large panel de programmes dans une version stable et éprouvée. Loin de se contenter d'un programme pour une tâche, le projet Debian laisse s'exprimer la diversité du logiciel libre en proposant à ses utilisateurs le plus grand choix possible, prêt à être installé d'une simple commande dès que vous en aurez besoin.

5 Autres services offerts par le projet Debian

Le projet Debian, au-delà des logiciels offerts à ses utilisateurs, assure de nombreux services pour mettre à jour vos systèmes, que ce soit pour leur sécurité ou pour vos propres besoins.

5.1 Sécurité de votre Wheezy

Plus que jamais, le service de sécurité fourni par le projet Debian assure l'intégrité et la pérennité de votre système d'exploitation et des logiciels associés. Afin de bénéficier des mises à jour de sécurité pour votre Debian Wheezy, il est nécessaire d'avoir la ligne suivante dans votre fichier **/etc/apt/sources.list** :

```
deb http://security.debian.org/ wheezy/updates main contrib non-free
```

Il vous suffit ensuite de mettre à jour votre liste des paquets, puis d'installer toutes les mises à jour en bloc :

```
# aptitude update && aptitude upgrade
```

Vous pouvez également affiner l'opération et n'installer qu'une mise à jour en précisant le nom du paquet, par exemple pour Iceweasel :

```
# aptitude update && aptitude install iceweasel
```

Vous pouvez être tenu au courant des différentes mises à jour de sécurité via la liste de diffusion *debian-security-announce* **[5]**.

Fig. 1 : backports.debian.org vous permet d'explorer la liste des paquets rétroportés.

5.2 Meilleure intégration du rétroportage d'applications

Le service de rétroportage (*backport*) d'applications empaquetées bénéficie d'un franc succès et les demandes de rétroportage sont de plus en plus nombreuses. Afin de rendre plus simple à l'utilisateur l'emploi de ce service permettant d'installer sur votre système stable des logiciels plus récents que ceux proposés par défaut dans le dépôt de la version stable, la syntaxe suivante sera désormais à employer dans votre fichier **/etc/apt/sources.list** pour déclarer le dépôt de rétroportage :

```
deb     http://ftp.fr.debian.org/debian wheezy-backports main contrib
deb-src http://ftp.fr.debian.org/debian wheezy-backports main contrib
```

Il vous suffit ainsi pour l'invoquer de préciser le dépôt des backports en utilisant **apt-get**. Nous souhaitons par exemple mettre à jour Samba sur notre système avec la version 3.6.15 rétroportée et donc plus récente que la 3.6.6 présente par défaut dans Wheezy :

```
# apt-get -t wheezy-backports install samba
```

Voir Figure 1.

5.3 Mises à jour stables pour Wheezy

Sans attendre les publications mineures de Wheezy qui seront publiées au compte-gouttes tout au long de la vie de cette version, il est possible d'accéder directement et sans attendre aux mises à jour d'antivirus, aux données des fuseaux horaires et de changements d'heure. Il suffit pour cela d'ajouter les lignes suivantes à votre fichier **/etc/apt/sources.list** :

```
deb http://ftp.debian.org/debian wheezy-updates main
deb-src http://ftp.debian.org/debian wheezy-updates main
```

De la même façon qu'avec les dépôts de rétroportage, vous devez mettre à jour votre liste de paquets avant de pouvoir installer la mise à jour, par exemple ici avec ClamAV :

```
# aptitude update && aptitude -t wheezy-updates install clamav
```

Conclusion

Le processus de publication de Wheezy a été fidèle à la devise du projet : *When it's ready.* Toutefois, les annonces du début de la période de gel et sur l'état d'avancement du travail de la réduction du nombre de bugs critiques ont été faites régulièrement et ont été largement relayées par les différents canaux officiels du projet, ainsi que par les membres de la communauté. Le temps de maturation de Wheezy a été sensiblement plus long que Squeeze (10 mois contre 6), mais le nombre de paquets proposés, les architectures et les services intégrés n'ont jamais été aussi divers et bien intégrés. La première mise à jour mineure a été publiée dès le 15 juin 2013, assurant la large diffusion des premiers correctifs développés depuis la sortie de Wheezy.

Avec cette version 7.0, le projet Debian s'affirme plus que jamais fidèle à ses principes, qui promeuvent le logiciel libre et le confort des utilisateurs de son système. ■

Références

[1] http://archive.debian.org/
[2] http://www.debian.org/releases/stable/amd64/release-notes/ch-upgrading.fr.html
[3] http://live.debian.net/
[4] http://qa.debian.org/developer.php?login=pkg-multimedia-maintainers@lists.alioth.debian.org
[5] http://lists.debian.org/debian-security-announce/

1&1 SERVEUR CLOUD DYNAMIQUE

PUISSANCE FLEXIBLE

PRIX MAÎTRISÉ

à partir de **0,03 €** HT PAR HEURE*

CONTRÔLE TOTAL DES COÛTS

- **NOUVEAU : sans engagement !**
- **NOUVEAU : sans frais de mise en service !**
- **NOUVEAU : sans prix de base !**
- **Durée limitée : jusqu'à 30 € de crédit offert !****
- **Transparence totale** grâce à la facturation horaire à l'usage.
- **Trafic illimité** sans réduction de bande passante.
- **Parallels® Plesk Panel 11 inclus,** noms de domaine illimités.

HAUTE FLEXIBILITÉ

- vCores, RAM et espace disque configurables séparément : **seulement 0,01 € HT par heure et par unité matérielle !***
- **NOUVEAU : jusqu'à 8 vCores et 32 Go de RAM.**
- Ajoutez jusqu'à 99 machines virtuelles en un seul clic - sans migration !

SÉCURITÉ OPTIMALE

- Disques durs et unités de calcul redondés afin de protéger votre serveur cloud contre toute défaillance.

ACCÈS ROOT COMPLET

- Droits d'administrateur et ressources dédiées pour chaque VM.

OPTERON PROCESSOR AMD · SUSE · Parallels Plesk Panel

1&1

MEMBRE DE united internet

DOMAINES | EMAIL | HÉBERGEMENT | E-COMMERCE | SERVEURS

0970 808 911 (appel non surtaxé)

1and1.fr

* Configuration minimale : 1 vCore, 1 Go de RAM et 100 Go d'espace disque, soit 0,03 € HT/heure (0,036 € TTC). Le prix final varie en fonction de la configuration choisie : simulation en ligne sur 1and1.fr/cloud-server-config

** Crédit de 30 € déduit du montant HT de la 1ère facture, pas de report du crédit non consommé. Détails disponibles sur 1and1.fr.

KERNEL 3.8 :

INFRASTRUCTURE DES ESPACES DE NOMS ET ESPACE USER

par Eric Lacombe

Nous traitons dans cet article de l'infrastructure des espaces de noms et présentons un aspect essentiel qui a été intégré au noyau 3.8, à savoir l'instantiabilité de l'espace des comptes utilisateur ou, autrement dit : la possibilité de créer de nouveaux espaces d'UID et GID.

Nous terminons sur l'analyse de quelques failles de sécurité ayant affecté ces espaces, permettant ainsi au lecteur d'entrevoir les risques que font peser sur la sécurité d'un système, l'introduction de fonctionnalités dont l'ensemble des combinaisons révèle une complexité dure à appréhender.

1 Introduction aux espaces de noms sous Linux

Qu'est-ce qu'un espace de noms pour Linux ? Il s'agit tout d'abord de comprendre qu'un système d'exploitation fournit différents types de ressources aux applications que l'on peut répartir en ressources d'exécution (CPU, GPU, etc.), ressources de mémoire (mémoire principale, mémoire de masse, etc.) et ressources de communication (réseau, IPC, affichage, etc.). Un des objectifs du système d'exploitation est de mettre à disposition ces ressources *à l'ensemble* des applications, ce qui impose donc qu'il soit capable de distinguer ces applications pour être à-même de distribuer les ressources (que cette distribution soit équitable ou non). Un espace de noms peut donc être vu comme la mise en œuvre de cette distinction des applications par rapport aux différentes ressources ; des noms deux à deux distincts (servant d'identifiants) étant associés aux différentes applications ou, plus précisément, aux différentes *tâches* s'exécutant sur le noyau Linux. Est appelée une tâche, la plus petite entité distinguable par le noyau Linux pour la distribution de ressources. On parle également de « fil d'exécution système » ou de *thread* noyau. A noter que ces noms expriment l'association de fait d'une tâche à la ressource d'exécution. Les tâches sont donc explicitement la représentation des entités atomiques pour le partage de ressources d'exécution. Elles sont également les entités que le noyau utilise pour la distribution des autres ressources. Bien que ce choix ait du sens dans la majorité des contextes, pour d'autres, il serait préférable d'avoir une vue différentes des entités en fonction du type de ressources. C'est en ce sens que l'on peut voir le service que rend l'infrastructure des *control groups*, permettant la répartition des ressources par groupe de tâches, ces groupes pouvant être différents d'un type de ressources à l'autre.

Nous venons de parler d'un espace de noms capital pour le noyau Linux, celui des identifiants des tâches (espace des PID). Mais quels sont alors ces autres espaces de noms ? Tout comme il faut distinguer les entités auxquelles ont fourni les ressources, il faut également distinguer les ressources elles-mêmes pour chaque type de ressources. D'où la notion d'espace de noms : de réseau, d'IPC, de système de fichiers, etc.

Nous allons voir dans la suite comment s'articulent ces différents espaces de noms entre eux. A noter que les informations fournies dans cet article sont valables à partir d'un noyau 3.8.

Les différents espaces de noms actuellement mis en œuvre dans le noyau sont les suivants :

`CLONE_NEWUSER` : déclenche la création d'un nouvel espace de comptes utilisateur (appelé « espace USER » par la suite). Nous détaillons cet espace dans la suite de l'article.

`CLONE_NEWPID` : déclenche la création d'un nouvel espace de PID. La création d'un nouvel espace de PID permet de fournir au nouveau processus créé, une vue indépendante sur les processus du système expurgée de tout processus qui existait préalablement. Le processus créé est alors placé à la tête du nouvel espace de PID, ce qui signifie que le noyau lui attribue le PID 1, correspondant au processus **`init`**. Une conséquence directe est que tout nouveau processus dans cet espace de nom, qui se retrouve orphelin, est alors placé sous le giron du processus ayant initié le nouvel espace de

PID. Notons que contrairement au vrai processus **init**, le processus initial d'un nouvel espace de PID peut se terminer et, dans ce cas, l'ensemble des processus de l'espace de PID est alors terminé. Remarquons finalement que les processus d'un nouvel espace de noms sont visibles depuis l'espace parent (l'inverse étant faux), ce qui implique l'existence possible de multiples PID pour un processus suivant les espaces de PID dans lesquels il est visible.

CLONE_NEWNET : déclenche la création d'un nouvel espace réseau, qui consiste ainsi en des instances différentes : des interfaces réseau, des piles protocolaires IPv4 et IPv6, des tables de routage IP, des règles de pare-feu, des arborescences **/proc/net** et **/sys/class/net**, des sockets, etc. À noter qu'une interface réseau physique ne peut être présente que dans un seul espace de noms. L'instance de cette interface dans d'autres espaces doit se faire alors au travers d'interfaces virtuelles (**veth**) associées à l'interface physique et agissant à la façon de **pipe**.

CLONE_NEWNS : déclenche la création d'un nouvel espace de montage, qui consiste à isoler l'ensemble des points de montage du système de fichiers pour le ou les processus résidants dans le nouvel espace. Ainsi, des processus disposants d'espace de montage différent peuvent avoir une vue différente du système de fichiers. A noter que la création d'un nouvel espace de noms n'implique pas la suppression des points de montage tels que vu par le processus parent. Plus précisément, lors de cette création, la structure **task->nsproxy->mnt_ns** contenant l'ensemble des informations relatives aux points de montage est copiée depuis le processus parent. Il est donc nécessaire de faire appel à **mount()** et **umount()** pour changer cette vue (ces appels système agissent seulement sur l'espace de montage du processus appelant). Notons aussi que l'utilisation de ce type d'espaces permet d'obtenir une isolation plus sécurisé sur le système de fichiers que via l'utilisation de **chroot()** (*cf.* sections 3.7.2 et 3.7.3 pour plus d'informations sur ce sujet). En outre, les espaces de montage fournissent plus de flexibilité à l'utilisateur pour créer son environnement. Il est en effet possible de créer des relations entre ces espaces de noms de façon à propager automatiquement des événements de montage d'un espace à l'autre (se référer par exemple aux options **--make-shared** et **--make-slave** de la commande **mount**).

CLONE_NEWIPC : déclenche la création d'un nouvel espace de ressources IPC (Inter Process Communication), qui consiste à isoler certaines ressources IPC, à savoir, les IPC System V et les files de messages POSIX.

CLONE_NEWUTS : déclenche la création d'un nouvel espace d'identifiants système, qui consiste à isoler deux identifiants du système, à savoir **nodename** et **domainname**, qui sont retournés par l'appel système **uname()**. Les appels système **sethostname()** et **setdomainname()** servent à positionner ces identifiants.

2 L'API des espaces de noms

Dans cette section nous décrivons les trois appels système qui composent l'API des espaces de noms. Il s'agit de **clone()**, **setns()** et **unshare()**. Le premier est loin d'être un appel système nouveau dans Linux ; il sert notamment à l'implémentation de l'appel POSIX **fork()**. Il est toutefois plus flexible et permet de créer un processus dans un nouvel espace de noms vis-à-vis des différents types de ressources supportés. L'appel système **unshare()** agit quant à lui sur un processus déjà existant et va lui permettre de se détacher des espaces de noms dont il a hérité de son père, créant alors de nouveaux espaces de noms. Enfin, **setns()** est utile pour déplacer un processus d'un espace de noms à un autre. Ces trois appels système sont les seuls disponibles pour interagir avec l'infrastructure des espaces de noms. Cette dernière présente toutefois des entrées dans **procfs**, révélant les espaces de noms de chaque processus du système, qui servent de liant aux appels systèmes que nous venons de mentionner. Il faut également ajouter les constantes : **CLONE_NEWIPC**, **CLONE_NEWNS**, **CLONE_NEWNET**, **CLONE_NEWPID**, **CLONE_NEWUSER**, **CLONE_NEWUTS**, qui sont communes à ces derniers et servent à distinguer les différents types d'espace de noms.

Chaque processus présente dans **procfs** les identifiants des espaces de noms dans lesquels il se trouve. Ces identifiants correspondent à des fichiers dans le répertoire suivant (pour un processus identifié par **<PID>**) :

```
/proc/<PID>/ns
```

Ce répertoire contient les fichiers suivants (sauf si les espaces de noms correspondant n'ont pas été configurés dans le noyau) :

```
$ ls -l /proc/self/ns/
total 0
lrwxrwxrwx 1 tuxico tuxico 0 juil. 23 00:09 ipc ->
ipc:[4026531839]
lrwxrwxrwx 1 tuxico tuxico 0 juil. 23 00:09 mnt ->
mnt:[4026531840]
lrwxrwxrwx 1 tuxico tuxico 0 juil. 23 00:09 net ->
net:[4026531956]
lrwxrwxrwx 1 tuxico tuxico 0 juil. 23 00:09 pid ->
pid:[4026531836]
lrwxrwxrwx 1 tuxico tuxico 0 juil. 23 00:09 user ->
user:[4026531837]
lrwxrwxrwx 1 tuxico tuxico 0 juil. 23 00:09 uts ->
uts:[4026531838]
```

Il s'agit de liens symboliques pointant sur des noms constitués du type d'espace de noms, suivi d'un numéro d'*inode*. Ces numéros d'inode servent à distinguer des espaces de noms différents. Ainsi des processus qui se trouvent dans des espaces de noms différents révèlent nécessairement des liens symboliques pointant vers des inodes différents. A noter que cette information peut être obtenue grâce à l'appel système **stat()**. Le noyau retourne la valeur de l'inode dans le champ **st_ino** de la structure renvoyée.

Lorsque l'on souhaite qu'un espace de noms persiste en mémoire, sans pour autant qu'un processus y soit associé, il est possible d'effectuer un « **mount --bind** » de son lien symbolique (dans **procfs**) vers un fichier quelconque du système de fichiers. Dès lors, la terminaison des processus contenus dans cet espace n'entraînera plus la destruction de ce dernier. Cette pratique peut faciliter la gestion des espaces de noms en y associant des noms clairement identifiés (c'est-à-dire qui ne varient pas avec le **PID** des processus) au niveau du système de fichiers qui persiste indépendamment du cycle de vie des processus y étant associés. Les commandes suivantes réalisent cela à partir de l'espace USER du processus 2555 :

```
# touch ~/user
# mount --bind /proc/2555/ns/user ~/user
```

```
int clone(int (*child_func)(void *), void
*child_stack, int flags, void *arg);
```

Cet appel système est la primitive de base fournie par le noyau Linux pour créer de nouvelles tâches. Elle sert ainsi à la création des nouveaux processus et des nouveaux *threads*. Son argument **flags** permet de paramètrer finement la création en spécifiant notamment les attributs que la nouvelle tâche doit partager avec son père (comme ses descripteurs de fichiers ouverts via le flag **CLONE_FILES**, sa mémoire via le flag **CLONE_VM**, etc.). Cet argument permet également de demander à placer la tâche en cours de création dans de nouveaux espaces de noms, grâce aux flags **CLONE_NEW***. (Notons que pour passer l'ensemble des flags qui nous intéresse à clone(), il faut en calculer le « OU » binaire.)

Enfin, l'argument **child_func** est la fonction que la tâche créé doit exécuter, **child_stack** est un pointeur vers un espace mémoire qui jouera le rôle de sa pile d'exécution et **arg** correspond à un pointeur sur un argument qui sera passer à la fonction **child_func()**.

```
int setns(int fd, int nstype);
```

Cet appel système permet à une tâche de rejoindre un espace de noms identifié par **fd** (tout en quittant celui dans lequel elle se trouvait avant). **fd** est un descripteur de fichier pointant sur le lien symbolique de l'espace de noms que l'on souhaite rejoindre. Il peut être obtenu directement en faisant appel à **open()** sur le lien symbolique, ou bien sur un fichier *bind mounté* sur ce lien.

nstype est pris en compte par le noyau uniquement s'il est égal à une des constantes **CLONE_NEW***. Dans ce cas il sert de moyen de vérification pour le noyau qui s'assure que l'espace de noms que l'on souhaite rejoindre est bien du type précisé. Si sa valeur est **0**, alors aucune vérification n'est faite.

```
int unshare(int flags);
```

Cet appel système permet à la tâche appelante de créer et de rejoindre un ou plusieurs nouveaux espaces de noms (en se séparant des anciens bien évidemment), lesquels doivent être spécifiés par les constantes **CLONE_NEW***. Plus précisément, le « OU » des constantes doit être passé en argument de la fonction. Contrairement à **clone()**, cet appel permet d'éviter la création d'une nouvelle tâche.

L'espace des **PID** a pour particularité d'être traité d'une façon différente par les appels **unshare()** et **setns()**. En effet, lorsqu'une tâche déclenche ces appels système avec le flag **CLONE_NEWPID**, elle n'est pas placée dans un nouvel espace de noms. Seuls ces futurs enfants seront créés à l'intérieur de ce nouvel espace. Ce traitement particulier s'explique par le fait que le **PID** d'une tâche ne doit pas changer durant toute la vie de celle-ci, car de nombreux programmes et bibliothèques font cette hypothèse. Tout particulièrement, la **glibc**, sur laquelle s'appuie la construction de nombre de ces programmes, implémente l'appel système **getpid()** en mettant en cache le résultat afin d'optimiser les futurs appels. On comprend alors que le respect de cet invariant est primordial pour éviter l'apparition de bugs dans l'espace utilisateur.

3 L'espace de noms sur les comptes utilisateur

3.1 Vue d'ensemble

La brique qui manquait à l'édifice des espaces de noms est intégrée depuis la version 3.8 au noyau Linux. Elle est d'une importance capitale, car elle permet enfin pour un utilisateur non-privilégié de créer des environnements cloisonnés, dans lesquels peuvent s'exécuter ses processus auxquels il peut fournir un accès aux fonctionnalités du système normalement réservées à root (tout en restant dans le cadre des environnements cloisonnés).

En effet, à la différence de tous les autres espaces de noms, celui-ci ne requiert aucun privilège pour être créé. Il fournit en outre au premier processus créé dans le nouvel espace de noms tous les privilèges dans ce dernier (tout en ne lui en associant aucun dans l'espace parent), lui rendant alors accessible la création d'autres espaces de noms et donc d'un environnement dans lequel il pourra se considérer comme un roi. Ce dernier point reste à nuancer car ce nouvel environnement ne pourra être enrichie que par des ressources auxquelles à accès l'**UID** dans l'espace parent de l'utilisateur ayant créé le nouvel espace.

Lors de la création d'un nouvel espace USER via **clone()** ou **unshare()** avec le flag **CLONE_NEWUSER**, la tâche créée dispose d'un **UID** et d'un **GID** égal à **65534**. Plus exactement, cette valeur correspond à ce qui est inscrit dans les entrées **/proc/sys/kernel/overflowuid** et **/proc/sys/kernel/overflowgid**, au moment de la création. Une caractéristique importante de cette nouvelle tâche est qu'elle dispose de toutes les capacités dans ce nouvel espace, lui fournissant des droits illimités dans cet environnement et donc l'accès aux fonctionnalités réservées habituellement à root. Toutefois, comme déjà mentionné, elle sera privée de toute capacité dans l'espace parent au moment de sa création et cela afin d'éviter de potentielles escalades de privilèges.

Cette nouvelle tâche peut en outre changer son **UID** et **GID** dans le nouvel espace, ou même laisser son père s'en charger. Pour cela il faut passer par les interfaces **/proc/<PID>/uid_map** et **/proc/<PID>/gid_map** qui permettent de réaliser une correspondance entre des **UID** de l'espace de noms créé et dans lequel se trouve la nouvelle tâche (d'identifiant **PID**) et l'espace de nom parent.

3.2 Écriture dans uid_map et gid_map

L'écriture dans les interfaces **/proc/<PID>/uid_map** et **/proc/<PID>/gid_map** doit respecter le format suivant :

```
ID-inside-ns   ID-outside-ns   length
```

avec :

- **ID-inside-ns** : l'UID à l'intérieur de l'espace de noms créé,
- **ID-outside-ns** : l'UID tel que vu depuis un autre espace de noms (cette valeur correspond par exemple pour

l'espace de noms parent à l'UID de son espace qui est mis en correspondance avec **ID-inside-ns**),

- **length** : la plage des UIDs à mettre en correspondance.

Au delà du respect de ce format, un certain nombre de règles doivent être respectées pour que l'écriture soit prise en compte par le noyau :

- La mise en place d'une correspondance entre **UID** / **GID** d'un espace de noms donné vers un autre espace ne peux se faire qu'une seule fois en une seule opération d'écriture sur les fichiers **uid_map** et **gid_map** d'un unique processus. Cette opération peut cependant contenir plusieurs lignes (jusqu'à cinq dans un noyau 3.8) ; afin d'établir des correspondances de plusieurs plages d'**UID** / **GID**.
- Les fichiers **/proc/<PID>/uid_map** et **/proc/<PID>/gid_map** ont pour propriétaire l'utilisateur qui a créé l'espace de noms et ne sont inscriptibles que par lui (ou par un utilisateur privilégié).
- Le processus réalisant l'écriture doit disposer des capacités **CAP_SETUID** et/ou **CAP_SETGID** dans l'espace de noms du processus **PID**.
- Le processus écrivain doit obligatoirement se trouver soit dans le nouvel espace de noms, soit dans l'espace de noms parent.
- L'une des deux conditions suivantes doit être vérifiée (et seulement une peut l'être de part la première règle) :
  - → Ce qui est écrit dans les fichiers **uid_map** et **gid_map** consiste en une seule ligne écrite par le premier processus créé dans le nouvel espace de noms. Cette ligne ne peut établir que la correspondance entre l'**UID** / **GID** effectif du processus dans l'espace de noms parent vers l'**UID** / **GID** dans le nouvel espace de noms.
  - → Pour établir de multiples correspondances entre les **UID** / **GID** de l'espace parent vers le nouvel espace, le processus écrivain doit disposer obligatoirement de la capacité **CAP_SETUID** / **CAP_SETGID** dans l'espace de noms parent. Cette opération ne peut donc être réalisée que par un processus de l'espace parent, car le processus initial créé dans un nouvel espace USER ne dispose d'aucun droit dans l'espace parent.

Notons également qu'il est possible de créer des espaces de noms imbriqués, pour lesquels la relation père-fils permet de préserver la cohérence dans la gestion des droits lors de l'accès aux ressources notamment. Par exemple, un fichier créé sur un disque avec un UID 0 dans un nouvel espace de noms, lequel est mappé à un UID 1000 dans l'espace parent, sera écrit sur disque avec un UID 1000, évitant de fait une faille de sécurité. Lorsque ce fichier est lu depuis l'espace enfant, il est par contre présenté comme ayant un UID 0.

3.3 Prise en compte des espaces USER dans le fonctionnement des capacités

L'interprétation des capacités des processus par le noyau a dû quelque peu évoluer pour prendre en compte les espaces de noms, on peut résumer cette évolution à ce qui suit :

1 - Un processus obtient une capacité donnée dans un espace USER si et seulement si il fait partie de cet espace USER et que la capacité fait partie de ses capacités effectives. Par exemple, un appel à **clone()** avec le flag **CLONE_NEWUSER** octroie au fils toutes les capacités ; de plus, ce fils étant créé dans le nouvel espace, il dispose alors de toutes les capacités dans cet espace.

2 - Un processus créant un nouvel espace USER, dispose de toutes ses capacités dans ce nouvel espace, ainsi que dans les potentiels futurs espaces descendants. Cela permet au processus « racine » des espaces d'avoir toujours un contrôle sur sa descendance.

3 - Lorsqu'un nouvel espace USER est créé, le noyau enregistre l'**UID** effectif du créateur comme propriétaire de l'espace USER. Lorsqu'un processus de même **UID** et résidant dans l'espace USER parent rejoint le nouvel espace, le noyau lui octroie toutes les capacités. De plus, la deuxième règle assure que ce processus dispose aussi de toutes les capacités dans les espaces descendants.

Il est à noter que ces règles qui régissent l'allocation des capacités dans les espaces USER sont centrées sur les relations de descendances uniquement. Une conséquence intéressante est qu'un processus d'un espace USER donné ne dispose pas de capacités dans un espace frère. C'est à dire que, si un processus crée différents espaces USER, ces derniers seront isolés entre eux.

Enfin, rappelons qu'un espace USER peut être créé par n'importe quel processus, donnant accès à sa progéniture, à l'intérieur de cet espace, à toutes les fonctionnalités réservées habituellement à root. Il est alors légitime de craindre des escalades de privilèges, via l'accès à des appels systèmes susceptibles de modifier l'environnement d'exécution. Toutefois, lors de la création d'un nouvel espace USER, toutes les opérations privilégiés ne sont pas toutes disponibles pour les processus privilégiés créés à l'intérieur. En effet, les opérations qui requièrent un accès à des ressources qui sont attachées à l'espace parent et qui ne sont pas « instantiables », ne sont possibles que si l'**UID** / **GID** dispose des privilèges nécessaires dans l'espace parent. Ainsi, les ressources qui ne sont pas encore sous le giron de l'infrastructure des espaces de noms se trouvent dans cette situation, ce qui rassure ainsi sur les failles de sécurité que l'on aurait pu craindre jusqu'alors. Parmi ces opérations qui ne deviennent pas accessibles au sein d'un nouvel espace USER, on distingue par exemple : le réglage de l'heure, l'augmentation des limites d'utilisation des ressources (via **setrlimit()**), le chargement des modules noyau, la modification des priorités des processus, etc.

3.4 Perception des espaces USER entre eux

Les interfaces **/proc/<PID>/uid_map** et **/proc/<PID>/gid_map** fournissent lors de leur lecture une vue sur les

correspondances des **UID** / **GID** qui dépendent de l'espace de noms depuis lequel elles sont lues. C'est la partie **ID-outside-ns** qui va changer en fonction de l'espace de lecture. Sa valeur, en effet, va refléter l'**UID** ou le **GID** (suivant l'interface lue) dans l'espace lecteur. Prenons un exemple pour illustrer cela.

Depuis un shell, nous créons (avec un compte d'**UID 1000**) à l'aide d'un programme que nous appelons **exec_inside_new_userns**, un nouvel espace de nom USER avec une première tâche (supposons de **PID 2444**) dont la correspondance entre l'**UID 1000** de l'espace parent et l'**UID** que lui a attribué le noyau dans son espace, à savoir **65534** (valeur par défaut dans **/proc/sys/kernel/overflowuid**) est modifié par son père pour l'associer à l'**UID 0** dans le nouvel espace. La tâche fille exécute ensuite un *shell*.

Les lignes de commandes suivantes illustrent cela :

```
$ id -u
1000
$ ./exec_inside_new_userns '0 1000 1' bash
$ echo $$
2444
$ cat /proc/2444/uid_map
         0       1000          1
$ id -u
0
```

A noter que le programme **exec_inside_new_userns** réalise ce que nous venons de décrire en effectuant un appel à **clone()** qui crée une nouvelle tâche dont la fonction est d'exécuter via **execve()** le programme dont le nom lui est passé en dernier argument, mais après que son père ait écrit dans le fichier **uid_map** du fils ce qui est passé en premier argument. L'implémentation en C est fournie dans la section 3.5 et reprend en majeure partie le code d'exemple sur ce sujet de Michael Kerrisk. Insistons sur le fait que l'opération de modification de l'**UID** par la nouvelle tâche est effectuée avant que celle-ci ne fasse appel à **execve()** et cela en raison d'une règle de sécurité qui vaut sous tout système Linux et qui empêcherait la modification de l'**UID** après l'**execve()**. En effet, lorsqu'un processus, ayant un **UID** différent de zéro, effectue un appel à **execve()**, toutes ses capacités lui sont enlevées.

Dans un autre shell, nous utilisons à nouveau le programme **exec_inside_new_userns**, mais cette fois ci nous associant au processus fils (que nous supposons être de **PID 2555**) l'**UID 200**, en le mettant en correspondance avec l'**UID 1000** de l'espace parent.

```
$ ./userns_child_exec -U -M '200 1000 1' \
-G '200 1000 1' bash
$ cat /proc/self/uid_map
       200       1000          1
$ id -u
200
$ echo $$
2555
```

Si, toujours depuis ce shell, nous visualisons l'**uid_map** du processus tournant dans l'autre shell, on observe bien une correspondance cohérente avec notre espace :

```
$ cat /proc/2444/uid_map
            0        200          1
```

Si on procède de même mais depuis l'autre shell, pour visualiser le processus d'**UID 200**, on obtient bien toujours un résultat cohérent :

```
$ cat /proc/2555/uid_map
          200          0          1
```

Enfin, depuis un troisième shell, pour lequel l'espace USER est le même que l'espace parent, la visualisation des **uid_map** des processus 2444 et 2555 révèle bien la correspondance unique à l'**UID 1000**.

```
$ cat /proc/2444/uid_map
         0       1000          1
$ cat /proc/2555/uid_map
       200       1000          1
```

3.5 Implémentation du programme exec_inside_new_userns

Le code fourni dans cette section reprend en majeure partie le code d'exemple de Michael Kerrisk.

Nous commençons par présenter la fonction **main()** du programme **exec_inside_new_userns**, avant de poursuivre avec le reste de l'implémentation. Le **main()** débute par la récupération des arguments passés au programme, le premier étant une liste d'associations d'**UID**, le second étant la commande à exécuter suivie de ses arguments.

```
#define _GNU_SOURCE
#include <sched.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <signal.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <errno.h>
#define errExit(msg)    do { perror(msg);
exit(EXIT_FAILURE); \
                        } while (0)
struct child_args {
    char **argv;
    int    pipe_fd[2];
};
#define STACK_SIZE (1024 * 1024)
static char child_stack[STACK_SIZE];    /*
Space for child's stack */
int
main(int argc, char *argv[])
{
    pid_t child_pid;
    struct child_args args;
    char map_path[PATH_MAX];
    flags = CLONE_NEWUSER;
    uid_map = argv[1];
    args.argv = &argv[2];
```

Un **pipe** est alors créé par le père comme moyen de synchronisation avec le processus fils qu'il créé par la suite. Il permet au fils d'attendre la notification du père au travers du **pipe** avant d'exécuter la commande passée en argument de **exec_inside_new_userns**. Avant de notifier son fils, le père se charge d'établir la ou les correspondances d'UID tels que précisé en premier argument sur la ligne de commande. A noter que la structure **child_args** contient d'une part (dans le champ **pipe_fd**) les descripteurs des deux bouts du **pipe** dont nous venons de parler et d'autre part (dans le champ **argv**) la commande à exécuter par le processus fils.

```
    if (pipe(args.pipe_fd) == -1)
        errExit("pipe");
```

Le fils est alors créé dans un nouvel espace USER via l'appel à **clone()**. La fonction exécutée par ce fils est **childFunc** que nous détaillons par la suite. Une pile d'exécution doit également être passée en argument. Elle est réservée statiquement en mémoire à l'adresse **child_stack** (le « **+ STACK_SIZE** » s'explique par le fait que la pile grandit dans le sens des adresses mémoires décroissantes). Après la création du fils, le processus père met

à jour, via l'appel à **update_map()**, la correspondance d'**UID** tel que précisé en argument du programme (et maintenant référencé par **uid_map**). Il poursuit avec la notification du fils pour que celui-ci exécute enfin la commande. Cette notification est réalisée en fermant le bout du **pipe** utilisé pour l'écriture, ce qui a pour effet la réception d'un **EOF** (*End Of File*) du côté du lecteur.

```
    child_pid = clone(childFunc, child_stack +
STACK_SIZE,
                                CLONE_NEWUSER |
SIGCHLD, &args);
    if (child_pid == -1)
        errExit("clone");
    snprintf(map_path, PATH_MAX, "/proc/%ld/uid_map",
                (long) child_pid);
    update_map(uid_map, map_path);
    /* Close the write end of the pipe, to signal to
the child that we
       have updated the UID and GID maps */
    close(args.pipe_fd[1]);
    if (waitpid(child_pid, NULL, 0) == -1)      /*
Wait for child */
        errExit("waitpid");
    exit(EXIT_SUCCESS);
}
```

La fonction exécutée par le processus fils dans le nouvel espace USER est détaillée ci-dessous. La première opération est de fermer le bout du pipe pour l'écriture de façon à bien recevoir l'**EOF** dans la lecture qui suit lorsque le père fermera à son tour ce bout là du **pipe** (il ne faut pas oublier que lors du **clone()**, chacun des processus dispose de sa propre copie du **pipe**). L'appel à **read()** réalisé sur la partie en lecture du pipe est bloquante, ce qui permet effectivement la synchronisation avec le père.

```
static int
childFunc(void *arg)
{
    struct child_args *args = (struct child_args *)
arg;
    char ch;
    close(args->pipe_fd[1]);
    if (read(args->pipe_fd[0], &ch, 1) != 0) {
        fprintf(stderr, "Failure in child: read from
pipe returned != 0\n");
        exit(EXIT_FAILURE);
    }
    /* Execute a shell command */
    execvp(args->argv[0], args->argv);
    errExit("execvp");
}
```

Détaillons enfin la fonction **update_map()**. Son rôle est d'établir des associations d'**UID** entre le nouvel espace USER et celui du père. Elle prend en arguments : la liste des associations d'**UID** (**char *mapping**) et le chemin du fichier **uid_map** du processus fils (**char *map_file**). Notons que pour faciliter l'utilisation du programme **exec_inside_new_userns**, la commande autorise le passage d'associations d'**UID** comme une liste dont le séparateur est la virgule. Ce séparateur est alors remplacé par des retours chariots au sein de la fonction.

```
static void
update_map(char *mapping, char *map_file)
{
    int fd, j;
    size_t map_len;
    map_len = strlen(mapping);
    for (j = 0; j < map_len; j++)
        if (mapping[j] == ',')
            mapping[j] = '\n';
    fd = open(map_file, O_RDWR);
    if (fd == -1) {
        fprintf(stderr, "open %s: %s\n", map_file,
strerror(errno));
        exit(EXIT_FAILURE);
    }
    if (write(fd, mapping, map_len) != map_len) {
        fprintf(stderr, "write %s: %s\n", map_file,
strerror(errno));
        exit(EXIT_FAILURE);
    }
    close(fd);
}
```

3.6 Combinaison de plusieurs espaces de noms

Un des intérêts majeurs de **CLONE_NEWUSER** est de permettre, à n'importe quel utilisateur du système, la création de *containers*, c'est-à-dire d'environnements constitués de nouveaux espaces de noms (dont la création requière les privilèges root ou plus exactement la capacité **CAP_SYS_ADMIN**), comme **CLONE_NEWPID**, **CLONE_NEWNS**, etc.

Il en résulte pour cet utilisateur, la possibilité d'« isoler » du reste du système des programmes qu'il souhaite exécuter mais auxquels il ne souhaite pas accorder trop de confiance. Cette isolation ne couvre toutefois pas tous les aspects du système. Notamment, les ressources telles que la mémoire ou la charge CPU ne sont pas couvertes. Ces aspects relèvent plutôt de l'infrastructure des *control groups* qui n'est malheureusement pas intégrés avec les espace de noms. Pour rendre possible le contrôle des ressources depuis un container, sans toutefois lui donner ce contrôle sur l'ensemble des processus du système, l'approche recommandée est d'effectuer un « **mount --bind** » de la partie pertinente de la hiérarchie des *control groups* dans le container. Notons que le projet **LXC** vise à faciliter la création de tels containers depuis l'espace utilisateur en s'appuyant sur les espaces de noms et les *control groups*.

Pour en revenir aux espaces de noms, leur combinaison permet donc de paramétrer de façon poussée la « vue » qu'ont les processus du système sur lequel ils s'exécutent. Cette création de différents espaces de noms peut se faire en séquence, après plusieurs appels à **unshare()**, ou d'une traite via un seul appel à **clone()**, en combinant les différents flags. Dans ce dernier cas, le noyau prend garde de créer au préalable le nouvel espace USER (si le flag est présent) dans lequel le futur processus en cours de création dispose de toutes les capacités. Il s'attaque alors aux autres espaces, évitant ainsi tout échec lors de la création. Pour illustrer cela, donnons l'exemple de l'appel suivant qui parvient à créer l'ensemble des espaces de noms demandés, à partir d'un compte utilisateur non-privilégié :

```
clone(child_func, stack_ptr,
          CLONE_NEWUSER | CLONE_NEWPID |
CLONE_NEWIPC | CLONE_NEWNS, arg);
```

3.7 Défauts de jeunesse (vulnérabilités)

Cette section a pour but de montrer la difficulté pour le noyau Linux de garantir des propriétés de sécurité sur le système dès lors que des modifications en son cœur interagissent avec de nombreux sous-systèmes et dont les effets de bords dû aux différentes combinaisons possibles deviennent difficiles à appréhender dans leur globalité. Par ailleurs, cette difficulté est d'autant plus critique à gérer dans le cas des espaces USER, car elle ouvre soudainement la boite de pandore à tous les utilisateurs. Il faut entendre par cela que ces espaces mettent à disposition de tous les utilisateurs, nombres de fonctions qui étaient jusqu'alors limitées à root (car capable notamment d'affecter l'environnement d'exécution de n'importe quel processus). Comme nous avons pu le voir, une réflexion de fond a été mené pour contenir les effets des fonctions privilégiées aux nouveaux espaces créés. Toutefois,

l'exercice reste difficile comme nous allons le constater dans la suite au travers de l'explication de trois failles de sécurité ayant affecté les espaces USER dans la version 3.8 du noyau. Très vite corrigées, ces vulnérabilités ne devraient se trouver dans aucune distribution actuelle car :

- ces différentes vulnérabilités n'affectent que des versions 3.8 récentes, elles sont corrigées dans la version 3.8.3. De plus, les versions 3.9 et postérieures ne sont pas affectées, tout comme les versions antérieures ;
- peu de distributions activent dans le noyau les espaces USER avec un noyau 3.8 étant donné que certains systèmes de fichiers comme XFS ne sont pas supportés conjointement avec cette option.

En dépit de ces défauts de jeunesse, il faut souligner la réactivité de la communauté noyau dans la correction des failles de sécurité. Étant donné l'impossibilité d'avoir un noyau sans faille, cet aspect se révèle être un atout inestimable.

3.7.1 Contournement de la protection en lecture seule des systèmes de fichiers

La vulnérabilité (CVE-2013-1957), découverte par Andrew Lutomirski, permet de contourner la protection en lecture seule d'un système de fichiers bind-monté de la sorte, en créant un nouvel espace USER ainsi qu'un nouvel espace de montage. Le premier processus dans le nouvel espace USER obtient par le noyau toutes les capacités, il peut alors avoir accès à la commande mount. L'opération de « re-mountage » en lecture/écriture du système de fichier est donc accessible et fonctionne avec l'implémentation originale des espaces USER dans le 3.8.

La correction apportée est d'empêcher un remontage (via un nouveau flag **MNT_LOCK_READONLY**) lorsqu'un « montage » est copié d'un espace de montages à un autre et que ce dernier est associé à un espace USER moins privilégié (autrement dit, si l'utilisateur qui a monté le système de fichiers initialement disposait de plus de privilèges que le créateur de l'espace de montage).

3.7.2 Nouveau type d'évasion d'un environnement chrooté

La vulnérabilité (CVE-2013-1956), mise au jour par Andrew Lutomirski, permet de s'évader d'un environnement *chrooté*, en créant un nouvel espace USER suivi d'un nouvel espace de montage.

Le problème soulevé par cette vulnérabilité est très proche de celui qui affecte **chroot()** lorsque l'utilisateur chrooté dispose des privilèges suffisants pour exécuter **chroot()** (c'est-à-dire qu'il a la capacité **CAP_SYSCHROOT**), lui permettant ainsi de s'évader. Ainsi, pour mieux comprendre le scénario d'exploitation de cette nouvelle faille, nous expliquons tout d'abord la technique classique d'évasion d'un chroot. Mais avant tout, rappelons ce qu'est une **dentry** (*directory entry*) afin de comprendre le fonctionnement de l'appel système **chroot()**. Il s'agit d'une structure de base utilisée par le noyau pour parcourir les chemins du système de fichiers. Elle contient notamment : une chaîne de caractères correspondant au nom d'un bout de chemin , un pointeur vers l'inode de ce fichier et un pointeur vers la **dentry** parente.

L'appel à **chroot()** ne fait que remplacer la **dentry** racine dans le descripteur du processus appelant (**task->fs->root** et **task->fs->rootmnt**) avec celle qui correspond au chemin passé en argument, elle ne change pas le répertoire de travail courant (**task->fs->pwd** et **task->fs->pwdmnt**) et n'a aucune mémoire ; elle écrase purement et simplement la **dentry** racine actuelle. Ainsi, si le chemin commence par un **'/'** la nouvelle **dentry** racine est employée comme point de départ. De plus, si lors du parcours du chemin, il rencontre un **'..'** alors qu'il se trouve déjà à la racine, il l'ignore tout simplement. Cependant, lors du parcours d'un chemin relatif (c'est-à-dire qu'il ne contient pas de '/' en tête du chemin), le noyau pars de la **dentry** pointant vers le répertoire de travail courant (**task->fs->pwd**). Et si ce répertoire courant est en amont de la racine, le noyau ne va jamais ignorer des chemins **'..'** car son seul moyen de vérification se fonde sur l'égalité de la **dentry** courante avec la dentry racine. Il est donc possible de remonter le système de fichier en effectuant des appels successifs à **chdir('..')**, si le répertoire de travail courant se situe en amont de la racine du chroot. Ainsi, lorsqu'on est root, on peut toujours s'évader d'un environnement chrooté même si le répertoire de travail courant est bien positionné à l'intérieur du chroot. En effet, il suffit alors de créer un nouveau répertoire et de se chrooter dedans (d'où le besoin d'être root, ou plus exactement de la capacité **CAP_SYSCHROOT**). Et vu que **chroot()** ne modifie pas le répertoire de travail courant, nous nous trouvons bien dans la situation décrite précédemment. Il nous suffit donc de remonter jusqu'à la racine et d'effectuer un dernier **chroot()** à ce niveau, pour écraser la **dentry** racine du dernier chroot avec la dentry de la véritable racine. C'est pourquoi, il ne faut jamais laisser un utilisateur avec **CAP_SYSCHROOT**, dans un chroot.

Bien que **chroot()** soit couramment utilisé comme un mécanisme de sécurité, il n'en a jamais eu la vocation. Aussi, comme nous venons de le voir, certaines précautions doivent être prise lors de son utilisation.

Dans la vulnérabilité qui nous intéresse, le problème se révèle être la combinaison de différents facteurs. D'une part, la création d'un nouvel espace USER octroie toutes les capacités au premier processus créé dedans, ce qui rend alors possible la création d'un nouvel espace de montage. D'autre part, lors de sa création la **dentry** racine ainsi que l'ensemble des informations dans **task->fs** sont écrasés. Il n'y a donc plus de trace de la **dentry** racine précédente. Bien que le répertoire courant soit également écrasé lors de la création du nouvel espace de montage, cela n'est pas suffisant pour bloquer l'attaque. En effet, il suffit de conserver un *descripteur de fichier* ouvert sur un répertoire amont (lequel ne sera pas supprimer après un appel à **unshare(CLONE_NEWNS)**) et d'effectuer un **fchdir()** dessus, pour se retrouver dans la situation que nous avons décrite précédemment.

L'exploit dont nous donnons une version simplifiée dans la suite, profite également du flag **CLONE_FILES** pour que le processus créé dans le nouvel espace USER partage sa table des descripteurs de fichiers ouverts avec son père. Ainsi, dans l'exploit suivant, le fils qui s'exécute dans

un nouvel espace USER, réalise l'attaque et laisse son père jouer avec le descripteur de fichiers représentant la racine.

```
int fn(void *unused)
{
  int i;
  int fd;
  fd = open("/", O_RDONLY | O_DIRECTORY);
  unshare(CLONE_NEWUSER);
  unshare(CLONE_NEWNS);
  fchdir(fd);
  close(fd);
  for (i = 0; i < 100; i++) {
    if (chdir("..") != 0) {
      warn("chdir");
      break;
    }
  }
  fd = open(".", O_PATH | O_DIRECTORY);
  _exit(0);
}

int main(int argc, char **argv)
{
  int dummy;
  int status;
  close(3);
  signal(SIGCHLD, SIG_DFL);
  clone(fn, &dummy, CLONE_FILES | SIGCHLD, 0);
  wait(&status);
  fchdir(3);
  close(3);
  execv(argv[1], argv+1);
  return 0;
}
```

A noter que la suppression de la vulnérabilité a consisté à interdire les espaces USER dans un environnement *chrooté*. Cela ne réduit pas pour autant le potentiel des espaces USER, car un chroot peut être remplacé par un nouvel espace de montage (**CLONE_NEWNS**).

3.7.3 Escalade de privilèges : obtention des droits root par combinaison de flags

L'*exploit* (**clown-newuser**), implémenté par Sebastian Krahmer, révèle une vulnérabilité des espaces USER (CVE 2013-1858) permettant l'obtention des droits root depuis un processus s'exécutant sans privilège particulier, sur un noyau 3.8 compilé avec l'option **CONFIG_USERNS**. Il se compose d'un programme s'exécutant de façons différentes en fonction du contexte dans lequel il est lancé. Ce contexte est en fait modifié au cours de l'exécution du programme, qui se ré-exécute après chacune des étapes de son scénario d'exploitation. Nous décrivons dans la suite le déroulement de l'exploitation.

La première étape du programme malveillant (l'*exploit*) est de repérer l'endroit où il se trouve dans le système de fichier, en effectuant un **readlink()** sur **/proc/self/exe** (ce fichier est un lien symbolique qui pointe sur le binaire du programme dont le PID correspond à celui du processus qui accède à **/proc/self**). Cela lui permet de se ré-exécuter par la suite. Il fait alors un appel à **fork()**. Dans le père, il scrute périodiquement l'exécutable de l'*exploit*, dans l'attente de conditions favorables, lesquelles sont réunies lorsque le fichier a changé de propriétaire pour root et a le bit Set-UID positionné. Ces conditions réunies, le père exécute alors le programme (lui-même donc) qui détecte ses nouveaux privilèges et exécute alors un shell, lequel dispose des droits root.

Les conditions favorables sont mises en place par le fils. Pour y parvenir, celui-ci commence par créer un répertoire qui lui servira d'environnement de chroot. Dans ce répertoire il établit un lien symbolique **lib64/ld-linux-x86-64.so.2** pointant vers lui-même (l'exécutable **clown-newuser**). Dans cet environnement le chargeur dynamique est donc modifié et pointe vers le programme malveillant. (Rappelons que le chargeur dynamique de Linux est exécuté au lancement de tout programme compilé dynamiquement, afin notamment de charger les bibliothèques qui lui sont nécessaires.) Il crée également dans ce dossier un lien vers l'exécutable Set-UID root **/bin/su**. L'idée est donc d'exécuter ce programme dans un environnement contrôlé où le chargeur dynamique est remplacé par le programme malveillant.

Le piège étant mis en place, le processus fils poursuit sa tâche en créant lui même un fils (via l'appel à **clone(..., CLONE_NEWUSER | CLONE_FS, ...)**) mais, cette fois-ci, dans un nouvel espace USER pour lui octroyer tous les droits dans cet espace et en lui partageant, grâce au flag **CLONE_FS**, certains de ses attributs relatifs au système de fichiers (la racine du système de fichiers, le répertoire de travail courant et le masque de création de fichiers - **umask**). Le petit fils a pour seul rôle d'effectuer un **chroot()** sur le dossier créé précédemment et contenant le piège. L'utilisation du flag **CLONE_FS** implique que le changement affecte également le processus père. Ainsi, lorsque le père exécute le programme **/bin/su** (après avoir détecté la terminaison du processus fils), celui-ci s'exécute dans l'environnement piégé. Cette exécution débute par la création d'un processus par le noyau et l'attribution des droits root (vu que le bit Set-UID du programme **/bin/su** est positionné et que ce programme appartient à root). Le chargeur dynamique est alors exécuté par le processus avec les droits root, ce chargeur étant en fait le programme malveillant. Il ne reste alors plus qu'une étape : faire en sorte d'obtenir les droits privilégiés à l'extérieur de l'environnement chrooté. Pour ce faire le programme malveillant, exécuté cette fois en tant que chargeur dynamique, change le propriétaire de **/lib64/ld-linux-x86-64.so.2** pour lui assigner root à la place et lui établit le bit Set-UID ; ce fichier étant toujours le lien vers le programme malveillant. Cette opération effectuée, le fils se termine. Le processus initial, que nous avons décrit au tout début, détecte donc qu'il dispose des droits root. Il exécute alors, pour finir, un shell donnant un accès complet sur le système à l'attaquant.

Remarquons que pour que ce scénario fonctionne, le programme malveillant doit nécessairement être compilé statiquement afin que le chargeur dynamique soit présent dans l'exécutable, étant donnée que celui de l'environnement chrooté est remplacé. Il faut également que la protection sur les liens physiques ne soit pas activé (c'est-à-dire que **/proc/sys/fs/protected_hardlinks** soit à 0). En effet, cette protection, introduite depuis le noyau 3.6 (et depuis longtemps disponible dans le *patchset* **grsecurity**), interdit un utilisateur de créer un lien physique sur un fichier, à moins que ce fichier ne lui appartienne. Toutefois, l'exploit pourrait être adapté pour éviter d'utiliser des liens.

La correction a consisté à interdire la combinaison **CLONE_NEWUSER** et **CLONE_FS** lors d'un appel à **clone()**. Mais également, à empêcher au travers de **setns()** et **unshare()** que deux processus puissent partager les mêmes attributs du système de fichiers, autrement dit partager leur **fs_struct** (présente dans chacun des processus - **task->fs**), dès lors que ces processus se trouvent dans des espaces USER différents. ■

GRAPHING, LOGGING ET MONITORING 2.0. ÉPISODE II :

VOS JOURNAUX AVEC LOGSTASH

par Benoît Benedetti
[Administrateur Système Linux @ Université de Nice Sophia Antipolis]

Après un premier article sur l'utilisation de Graphite[GLMF156] pour générer vos graphiques, nous allons parler gestion de logs avec Logstash. La gestion des logs est un véritable casse-tête. Dispersés sur vos différents serveurs, il faut commencer par centraliser tout ça. Vient ensuite l'exploitation de tous ces messages, chaque application ayant un format différent, plus ou moins exotique. Chacun de ces formats est trop complexe pour être facilement traité directement par l'œil humain, et trop verbeux pour être facilement traité par des scripts et autres outils. Sans aborder la théorie de la journalisation[10COMMANDEMENTS], voyons plutôt comment Logstash[LOGSTASH] va vous aider dans le traitement de vos logs.

1 Présentation

Logstash est un framework de gestion de logs. On peut le décrire comme l'équivalent d'un tube (le fameux pipe | que les amateurs de ligne de commandes connaissent), mais un tube réseau dopé aux hormones : Il reçoit des messages en entrée, les filtre et les modifie au besoin, puis les envoie sur la destination de votre choix.

L'intérêt de Losgstash est de convertir les messages dans un format commun à leur réception, représenté en interne sous la forme d'un Hash. Les filtres vous donnent la possibilité de manipuler ce Hash. Le message est ensuite envoyé vers la sortie de votre choix, son contenu indexé, sur lequel il est plus simple de faire des recherches, une fois stocké.

Une infrastructure Logstash est généralement composée d'un *Shipper* (client installé sur une machine qui traite et envoie les logs), un *Broker* qui reçoit voire transfère les logs (offrant la possibilité de créer un cluster de brokers pour de la haute disponibilité), un *Indexer* qui met les messages dans un format commun et les indexe dans un *stockage*, et enfin une *interface de visualisation*.

Figure 1 : logo

Tous ces composants ne sont pas obligatoires pour utiliser Logstash, mais c'est une infrastructure typique, qui peut être répartie sur différents serveurs, et utiliser d'autres outils en complément de Logstash.

Cette flexibilité est permise grâce à de nombreux plugins d'entrée et de sortie, qui vont vous permettre d'insérer sans peine Logstash dans votre architecture. Ainsi, un des plugins d'entrée permet d'utiliser (r)syslog(-ng) comme Shipper depuis vos machines vers un serveur central Broker/Indexer Logstash. Des plugins de sortie vous permettront depuis ce serveur central, d'envoyer les messages vers un serveur de stockage et de recherche ElasticSearch, et/ou des outils de monitoring comme Nagios ou Graphite.

Nous allons commencer par installer et découvrir Logstash depuis un seul et même serveur, qui servira de Shipper

de ses propres logs vers lui-même, les indexera et stockera en local. Nous verrons ensuite quelles stratégies s'offrent à nous pour envoyer les logs depuis d'autres machines de votre réseau vers ce serveur central.

2 Installation

Logstash est compatible Linux, et nous utiliserons Debian comme distribution de référence. Disponible en version 1.1.9, c'est un programme JRuby, distribué sous forme de jar. Vous pourrez donc l'exécuter facilement, à l'aide de Java (en version 6 minimum), préalablement installé :

```
$ sudo aptitude install -y default-jre-headless
```

Dans la suite, j'utiliserai le répertoire **/opt** pour héberger les différents fichiers utiles à son exécution, dans une sous-arborescence de dossiers :

```
$ sudo mkdir -p /opt/logstash/{bin,conf,logs,run}
```

On peut maintenant récupérer l'archive jar de Logstash :

```
$ LOGSTASH_VERSION="1.1.9"
$ sudo wget https://logstash.objects.dreamhost.com/release/logstash-
${LOGSTASH_VERSION}-monolithic.jar -O /opt/logstash/bin/logstash-
${LOGSTASH_VERSION}-monolithic.jar
$ sudo ln -s /opt/logstash/bin/logstash-${LOGSTASH_VERSION}-
monolithic.jar /opt/logstash/bin/logstash.jar
```

3 Utilisation

Logstash possède deux arguments **[ARGUMENTS]** : **agent**, qui traite les événements et **webui**, pour lancer une interface web de visualisation. Nous allons commencer par utiliser le mode agent, qui est le cœur de Logstash. Nous verrons plus loin comment visualiser les événements via une interface web.

3.1 Configuration de l'agent

Logstash, comme indiqué précédemment, reçoit des messages en entrée, éventuellement leur applique un voire plusieurs filtres, puis les renvoie sur la sortie de votre choix (du moins, suivant un plugin output disponible). Un fichier de configuration de Logstash, contiendra donc au plus les 3 blocs de définitions suivants, au format YAML **[CONFIGURATION]** :

```
input {
 #Récupération
}
filter {
#Filtres à appliquer au besoin
}
output {
#Sortie
}
```

Une configuration comprenant au minimum un **input** et un **output**, les filtres étant optionnels.

En mode agent, Logstash peut utiliser un ou plusieurs fichiers de configuration avec l'option **-f**. C'est la méthode la plus utilisée bien sûr pour configurer le comportement de Logstash. Mais pour débuter et tester rapidement Logstash, nous allons utiliser l'option **-e**, qui permet de passer la configuration sur la ligne de commandes. Commande qui sera de la forme :

```
$ sudo java -jar /opt/logstash/bin/logstash.jar  agent  -e
'input{#entree} filter{#filtre} output{#sortie}'
```

En utilisant le plugin d'input **stdin** et celui d'output **stdout**, et en omettant les filtres que nous verrons plus tard, cela nous donne la commande :

```
$ sudo java -jar /opt/logstash/bin/logstash.jar  agent  -e 'input{
stdin{type => "entree-standard"} } output{ stdout{} }' -v
...
All plugins are started and registered. {:level=>:info}
```

Lancez donc cette commande, avec l'option optionnelle **-v**, car Logstash est assez long à démarrer. Cette option de verbosité permet de savoir quand Logstash est prêt (le message *All plugins are started and registered***.** apparaît dans la console).

Comme plugin d'entrée, nous avons utilisé **stdin**, qui permet d'envoyer des messages depuis l'entrée standard à Logstash. Ce plugin possède beaucoup de paramètres **[STDIN]**, mais un seul est obligatoire : c'est le paramètre **type**, qui, comme son nom ne l'indique pas et peut prêter à confusion, donne un label à tous les messages que Logstash recevra via cette entrée pour pouvoir traiter des messages avec un label particulier dans les filtres ou les plugins d'output. Le plugin d'output utilisé est lui aussi le plus simple disponible. C'est **stdout**, qui renvoie le message traité sur la sortie standard, et peut être utilisé sans paramètres **[STDOUT]**. Avec Logstash lancé, tapez un message quelconque dans la console :

```
On teste un message dans la console
2013-03-04T21:22:03.973Z stdin://debian/: On teste un message dans la console
```

Logstash nous affiche le message en l'ayant formaté au minimum(ligne 2 du listing), formatage de la forme :

```
Temps Source Message
```

Vous remarquez que l'heure affectée par défaut à notre message est l'heure à laquelle il a été traité, convertie au format UTC. Si vous voulez voir exactement comment Logstash transforme en interne un message en un Hash, relancez Logstash (après avoir fait [Ctrl]+[C]), puis passez l'option **debug => true**, au plugin **stdout** :

```
$ sudo java -jar /opt/logstash/bin/logstash.jar  agent  -e 'input{
stdin{type => "entree-standard"} } output{ stdout{debug=>true} }' -v
...
All plugins are started and registered. {:level=>:info}
Un autre message de test dans la console
{"@source"=>"stdin://debian/", "@tags"=>[], "@fields"=>{}, "@
timestamp"=>"2013-03-04T21:27:45.358Z", "@source_host"=>"debian",
"@source_path"=>"/", "@message"=>"Un autre message de test dans la
console", "@type"=>"entree-standard"}
```

Vous voyez en détail qu'une fois un message reçu, Logstash crée un Hash composé de plusieurs champs (**@source**, **@tags**, etc.). **@timestamp** par exemple, est la date donnée par Logstash à l'événement traité. Tous ces champs vont permettre de faire des recherches précises sur les messages, une fois ceux-ci stockés, comme nous le verrons plus loin.

Votre configuration de Logstash va vite devenir plus complexe que notre simple exemple, et le passage de celle-ci sur la ligne de commandes par l'option **-e** va s'avérer contraignant. Commencez par sauvegarder cette configuration basique, utile au débogage, dans un fichier **/opt/logstash/conf/output-debug.conf** :

```
input {
        stdin { type => "entree-standard" }
}

output {
        stdout { debug => true }
}
```

Vous pourrez utiliser ce fichier de configuration avec l'option **-f** :

```
$ sudo java -jar /opt/logstash/bin/logstash.jar  agent  -f /opt/
logstash/conf/output-debug.conf -v
```

Envoyer des messages depuis l'entrée standard, c'est bien pour les tests, les récupérer depuis vos fichiers de logs c'est déjà plus utile et un cas d'utilisation plus pratique de Logstash. Pour cela, on utilise le plugin d'input **file**, qui permet d'auditer un fichier pour récupérer tous les nouveaux messages, à la manière de la commande **tail**. Par exemple, créez un fichier **/opt/logstash/conf/input-syslog-file.conf** pour récupérer les logs du fichier **/var/log/syslog** :

```
input{
 file {
  type => "syslog"
  path => "/var/log/syslog"
 }
}
```

Vous voyez que nous avons donné le type (le label) **syslog** à tous les messages reçus sur cette entrée. Nous n'avons pas indiqué de bloc output, car nous allons lancer Logstash en passant le dossier **/opt/logstash/conf/** en paramètre de l'option **-f**, pour que tous ses fichiers soient utilisés comme configuration de Logstash (nous aurions pu inclure toute la configuration dans un seul fichier bien sûr, mais l'éclatement en plusieurs fichiers permet de maintenir plus facilement vos différents plugins d'input, de filtres et d'output). Ainsi, le plugin d'output **stdout** de **/opt/logstash/conf/output-debug.conf** sera chargé. Ce plugin d'output n'ayant pas défini de label via l'option **type**, il s'applique donc à tous les messages transitant par Logstash, quel que soit leur label affecté via **type**, et renverra donc tous les messages vers **stdout**. Lancez donc Logstash :

```
$ sudo java -jar /opt/logstash/bin/logstash.jar  agent  -f /opt/
logstash/conf/ -v
```

Plutôt que d'attendre qu'un message apparaisse dans **/var/log/syslog**, générez vous-même un message avec **logger**, depuis une autre console :

```
$ sudo logger -t MaCommandeImaginaire[512] "Ceci est un test''
```

Le message suivant devrait apparaître dans la console depuis laquelle est lancée Logstash :

```
{"@source"=>"file://debian/var/log/syslog", "@tags"=>[], "@
fields"=>{}, "@timestamp"=>"2013-03-04T21:35:16.171Z", "@source_
host"=>"debian", "@source_path"=>"/var/log/syslog", "@message"=>"Mar
4 22:35:15 debian MaCommandeImaginaire[512]: Ceci est un test", "@
type"=>"syslog"}
```

On retrouve notre message au format si particulier. D'ailleurs, un paramètre reste désespérément vide depuis le début de nos tests. Il s'agit du paramètre **@fields**, manipulable grâce aux filtres.

3.2 Les filtres

Les filtres permettent de manipuler les messages traités par Logstash. Ils sont particulièrement intéressants pour enrichir le champ **@fields** des messages stockés, pour pouvoir encore effectuer des recherches plus précises. Nous n'allons pas faire un tour exhaustif des filtres, Logstash en possédant plus d'une vingtaine.

Le filtre le plus courant est le filtre grok, qui permet d'analyser et de structurer à sa guise un message reçu. Revenons à notre message forgé de log cron précédent. Il serait intéressant d'exploiter celui-ci pour extraire des informations comme la machine source, ou la commande exécutée. C'est ce que permet entre autres le filtre grok, dans l'exemple de configuration ci-dessus, à sauvegarder dans **/opt/logstash/conf/filter-syslog.conf** :

```
filter {
  grok {
   type => "syslog"
   pattern => "%{SYSLOGLINE}"
  }
}
```

Via son option **pattern**, le filtre grok va analyser la variable **@message** du message reçu, et extraire des informations pour remplir la variable **@fields**, suivant un motif. Un motif grok est de la forme **REGEXP:nom_de_variable:**

lorsqu'une chaîne de caractères du message analysé correspond à **REGEXP**, il récupère la valeur de cette chaîne, et la stocke dans **@field['nom_de_variable]'**. Et là vous me demandez en quoi **SYSLOGLINE** est un motif grok, il n'est pas de la forme **REGEXP:nom_de_variable** ?

Logstash vient en fait par défaut avec un certain nombre de motifs les plus courants. Ceux-ci sont visibles dans le dossier **patterns** de son code source **[PATTERNS]**. Notre motif **SYSLOGLINE** est défini dans le fichier **linux-syslog** de ce dossier, à l'aide d'autres motifs prédéfinis :

```
...
SYSLOGLINE %{SYSLOGBASE2} %{GREEDYDATA:message}
```

Notre motif **SYSLOGLINE** définit que la chaîne de caractères composant une ligne syslog sera à décomposer en deux sous-chaînes par les motifs **SYSLOGBASE2** et **GREEDYDATA**. **GREEDYDATA:message** signifiant que la sous-chaîne correspondant au motif GREEDYDATA sera assignée à **@field['message']**. La définition du motif **SYSLOGBASE2** se trouve au début du fichier précédent:

```
SYSLOGBASE2 (?:%{SYSLOGTIMESTAMP:timestamp}|%
...
```

Motif qui inclut lui-même un autre motif **SYSLOGTIMESTAMP**, et ainsi de suite. **GREEDYDATA** n'inclut plus de motif, il est directement défini par une expression régulière :

```
GREEDYDATA .*
```

Pour connaître la définition de tous les motifs par défaut, il faut lire le fichier des motifs de base du code source **[GROK-PATTERNS]**. Les multiples inclusions de motifs consécutives permettent de faciliter leur lecture, ainsi que leur écriture si vous voulez ajouter vos propres motifs. Il existe une application en ligne **[GROKDEBUG]** pour vous aider à tester vos motifs sur un message (Figure 2).

Mais revenons à notre message syslog. Relancez Logstash qui va maintenant utiliser le nouveau fichier **/opt/logstash/filter-syslog.conf**, et générez à nouveau un message avec **logger** dans une autre console :

```
$ sudo java -jar /opt/logstash/bin/logstash.jar  agent  -f /opt/
logstash/conf/ -v
{"@source"=>"file://debian/var/log/syslog", "@tags"=>[], "@
fields"=>{"timestamp"=>["Mar  4 22:55:01"], "logsource"=>["debian"],
"program"=>["MaCommandeImaginaire"], "pid"=>["512"],
"message"=>["Ceci est un test"]}, "@timestamp"=>"2013-03-
04T21:55:02.125Z", "@source_host"=>"debian", "@source_path"=>"/
var/log/syslog", "@message"=>"Mar  4 22:55:01 debian
MaCommandeImaginaire[512]: Ceci est un test", "@type"=>"syslog"}
```

Notre filtre grok a décomposé le message pour en extraire les différentes chaînes demandées dans des variables de **@fields**, comme **program** ou **logsource**, qui vous permettront de faire des recherches précises sur les messages récupérés.

On peut remarquer que notre message contient la date à laquelle il a été produit, et que grok a extrait cette date

Figure 2

dans le paramètre **timestamp** du Hash **@fields**. Variable à ne pas confondre avec **@timestamp**, qui est la date définitive que Logstash attribue au message, qui lui donne par défaut celle à laquelle il l'a traité. Dans notre exemple, cela se traduit par une différence infime de quelques dixièmes de secondes. Rien de dramatique, mais dans certains cas cela peut être fâcheux. Prenons par exemple le cas du traitement de logs anciens : vous avez un Logstash nouvellement installé dans votre infrastructure de traitement de logs, et vous voulez lui injecter le contenu de vos anciens journaux système. Voyons comment utiliser les filtres pour modifier le champ **@timestamp** qui est censé indiquer la date du message, à partir du champ **@field['timestamp']** récupéré depuis le contenu du message.

La manière la plus simple d'injecter à Logstash le contenu de fichiers de logs depuis une machine distante, est d'écouter sur un port de notre serveur grâce au plugin d'input **tcp**, en rajoutant la configuration suivante au fichier **/opt/logstash/conf/input-syslog-tcp.conf** :

```
input {
 tcp {
  type => "syslog"
  port => 514
 }
}
```

Cette configuration écoute sur le port TCP 514, et applique à nouveau à ces messages le label syslog, pour réutiliser le filtre grok précédent. Il ne nous reste plus qu'à injecter le

contenu d'un fichier de logs anciens depuis la machine distante, à l'aide de **netcat** :

```
machinedistante$ sudo cat /var/log/messages | nc localhost 514
```

Ou, plus simplement, pour les besoins de notre exemple, on injecte un faux message avec une date volontairement ancienne depuis une autre console :

```
$ echo "Jan  1 18:11:28 debian MaCommandeImaginaire[512]: Ceci est un test" | nc localhost 514
```

Dans la fenêtre exécutant Logstash :

```
{"@source"=>"tcp://127.0.0.1:41067/", "@tags"=>[], "@fields"=>{"timestamp"=>["Jan  1 18:11:28"], "logsource"=>["debian"], "program"=>["MaCommandeImaginaire"], "pid"=>["512"], "message"=>["Ceci est un test"]}, "@timestamp"=>"2013-03-04T22:18:37.480Z", "@source_host"=>"127.0.0.1", "@source_path"=>"/", "@message"=>"Jan  1 18:11:28 debian MaCommandeImaginaire[512]: Ceci est un test\n", "@type"=>"syslog"}
```

Grok a à nouveau bien extrait la date réelle du message dans **@field['timestamp']**. Ne reste plus qu'à modifier **@timestamp** en conséquence. Le filtre **date** est prévu à cet effet. Rajoutez-le comme suit dans **/opt/logstash/conf/filter-syslog.conf**, après le filtre grok (les filtres sont évalués et exécutés dans l'ordre de leur apparition dans le fichier de configuration) :

```
filter {
...
  date {
    type => "syslog"
    match => [ "timestamp", "MMM dd HH:mm:ss", "MMM  d HH:mm:ss" ]
    locale => "US"
  }
}
```

Ici, on indique que ce filtre s'appliquera aux événements avec un label **syslog**. L'option match prend comme premier paramètre le nom du champ de **@fields** à utiliser pour récupérer la date (ici, **timestamp** pour **@fields['timestamp']**) et à attribuer à l'événement dans **@timestamp**. Suivi d'un ou plusieurs formats de date, compatibles Joda **[JODA]**, bibliothèque Java utilisée pour parser la date. Ici, on indique deux formats de dates possibles : les journaux syslog peuvent avoir la date du jour sur deux chiffres (**'dd'**) comme **'21'**, ou un seul (**' d'**) comme **' 2'**. On relance Logstash et on envoie à nouveau notre message via **netcat** :

```
{"@source"=>"tcp://127.0.0.1:41069/", "@tags"=>[], "@fields"=>{"timestamp"=>["Jan  1 18:11:28"], "logsource"=>["debian"], "program"=>["MaCommandeImaginaire"], "pid"=>["512"], "message"=>["Ceci est un test"]}, "@timestamp"=>"2013-01-01T17:11:28.000Z", "@source_host"=>"127.0.0.1", "@source_path"=>"/", "@message"=>"Jan  1 18:11:28 debian MaCommandeImaginaire[512]: Ceci est un test\n", "@type"=>"syslog"}
```

Et notre message a bien été daté suivant la date contenue dans le message, **@timestamp** étant équivalent à **@fields['timestamp']**.

Note

Pourquoi force-t-on la locale à **US** dans la configuration du filtre **date** ? Si vous lisez ce magazine, il y a de fortes chances que vous soyez francophone et vos machines configurées comme telles, avec **LANG=fr_FR.UTF-8**. Or, pour parser une date suivant l'expression donnée (ici **MMM dd HH:mm:ss**), la librairie Java Joda, avec une locale en français, s'attend à avoir des dates en français, ce qui n'est pas le cas des logs, qui ont des dates anglo-saxonnnes (January, February, etc.). On force donc la locale en anglais de Joda pour correspondre aux logs.

Voilà pour une introduction aux filtres. Sachant que je vous parle d'indexation et de recherche facilitées par l'utilisation de nos filtres depuis tout à l'heure, il serait peut-être tant de voir en pratique comment profiter de ce travail maintenant.

4 Stockage et recherche dans ElasticSearch

4.1 Configuration

Il existe plusieurs plugins d'output. Nous avons vu **stdout**, pour envoyer les messages traités par Losgatsh sur la sortie standard. Pratique, pour tester et debugger votre configuration. Mais en production vous préférerez envoyer vos messages vers un serveur de stockage et/ou un de vos outils de métrologie ou de graphique. Pour ces derniers cas, on trouve des plugins pour nagios, zabbix, opentsdb, ganglia ou encore graphite et **statsd**, pour ne citer que du libre, car l'adoption de Logstash a vu apparaître de nombreux plugins d'output vers des solutions tiers de métrologies payantes et propriétaires en mode Saas.

Pour ce qui est du stockage, cela va du simple plugin output **file**, qui écrit les messages en sortie sur un fichier disque, jusqu'au support de bases de données NoSQL comme MongoDB, en passant par Redis et Riak. Le plugin d'output le plus intéressant reste ElasticSearch, qui va nous permettre de stocker, indexer et parcourir facilement les événements traités par Logstash.

ElasticSearch est un indexeur Open Source, RESTFul, basé sur Apache Lucene. Logstash permet d'utiliser facilement ElasticSearch, car l'archive **jar** vient avec une instance embarquée, pour les utiliser conjointement rapidement. Pour se servir de cette instance embarquée, il faut logiquement utiliser le plugin d'output elasticsearch, via la création du fichier de configuration **/opt/logstash/conf/output-es.conf** suivant :

```
output {
  elasticsearch {
    embedded => true
  }
}
```

Ce plugin possède plusieurs autres options. En particulier pour utiliser votre propre instance ou un cluster ElasticSearch externe et autonome.

Relancez Logstash, patientez, une instance locale d'ElasticSearch devrait être démarrée parallèlement à Logstash, et va recevoir tous les messages traités (tous les messages, car comme précédemment, l'option **type** n'a pas été utilisée sur ce plugin). Vous pouvez vérifier les ports ouverts :

```
$ sudo ss -lpt | grep java
0   50  :::9200   :::*   users:(("java",16489,64))
0   50  :::9300   :::*   users:(("java",16489,35))
0   50  :::9301   :::*   users:(("java",16489,83))
```

Le port 9301 est le port utilisé par l'agent Logstash pour discuter avec l'instance ElasticSearch. Le port 9300 est le mode transport de ElasticSearch, qui permet de discuter entre instances ElasticSearch dans le cadre d'un cluster. Le port 9200 est utilisé pour communiquer avec l'instance ElasticSearch en HTTP.

Par défaut, l'instance embarquée d'ElasticSearch stocke ses données dans un dossier *data*, à la racine du dossier personnel de l'utilisateur exécutant Logstash. Autrement dit dans **/root/data** ici. Si vous désirez modifier ce comportement, ainsi que d'autres options d'ElasticSearch, il faut créer un fichier de configuration, toujours à la racine de l'utilisateur. Créez donc le fichier **/root/elasticsearch.yml** suivant :

```
path:
  logs: /var/log/elasticsearch.log
  data: /opt/logstash/data
```

ElasticSearch peut être interrogée via son API HTTP **[ESAPI]** et il existe une gemme Ruby **[GEM]** qui vous permettra de faire des recherches en ligne de commandes sur les données. Mais, avouons-le, vous n'aurez pas le confort d'utilisation d'une interface web. Sujet que nous allons aborder, après avoir configuré Logstash en tant que service.

4.2 Remarque

ElasticSearch est vraiment pensée comme solution d'indexation, et non de stockage. Surtout qu'un message au format texte dans un fichier prend moins de place que son équivalent indexé dans ElasticSearch. Notre exemple utilise seulement une instance ElasticSearch comme output. Je vous invite à rajouter un plugin d'output **file**, pour stocker les messages au format texte dans un fichier. Solution de stockage à préconiser sur le long terme, surtout pour les logs dont la politique de conservation vous demande de les stocker sur une longue période. ElasticSearch vous servira pour les recherches et l'observation des tendances de vos logs à court et moyen termes. Rendez-vous sur le wiki de Logstash **[WIKI]** pour l'utilisation de templates **[ESTPL]** sur votre instance d'ElasticSearch, et autres commandes pour optimiser les index. Vous pourrez aussi supprimer les indices suivant leur ancienneté, à l'aide d'un script **[ESCLEAN]**.

5 Logstash as a service

Lancer Logstash en ligne de commandes, c'est pas très pratique. Nous allons créer un fichier de script init pour gérer Logstash en tant que service. J'ai mis un tel fichier, long, à disposition sur http://sebsauvage.net/paste/ **[LOGSTASH.INIT]**, le service de sebsauvage à la Gist, s'appuyant sur son projet libre ZeroBin. Une fois le contenu de la note précédente sauvegardée dans un fichier **/etc/init.d/logstash**, donnez-lui les droits d'exécution.

```
$ sudo chmod u+x /etc/init.d/logstash
```

Avant de lancer le service, supprimez du dossier de configuration (ou déplacez pour utilisation ultérieure), le fichier qui utilise les plugins **stdin** et **stdout** :

```
$ sudo  mv /opt/logstash/conf/output-debug.conf /opt/logstash/
```

Un service étant démarré en arrière-plan, d'où l'inutilité de tels plugins dans ce mode. Puis, démarrez Logstash et

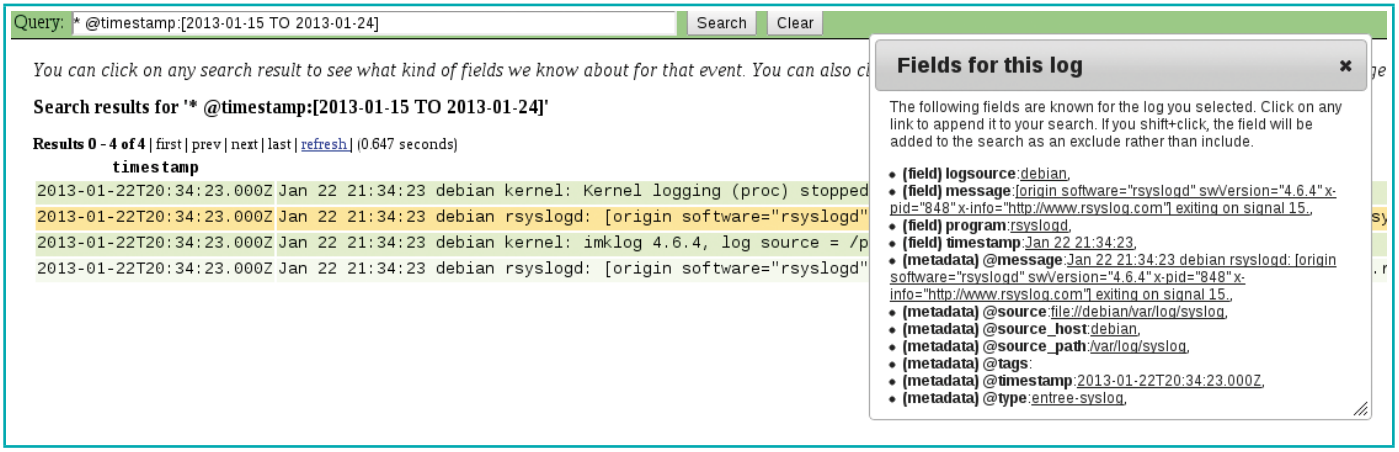


Figure 3

patientez, car le démarrage sera toujours un peu long, même démarré en arrière-plan :

```
$ sudo /etc/init.d/logstash start
```

6 Visualiser vos Logs avec Kibana

6.1 La Web UI officielle

Logstash est livré avec une interface web pour pouvoir naviguer et rechercher dans vos logs. Cette interface a l'avantage d'être simple à mettre en place, via l'option **webui**. Cette interface est un peu trop minimaliste (Figure 3 page précédente), et les recherches via la barre supérieure peu ergonomiques. De l'aveu du développeur de Logstash, qui n'a pu se focaliser plus sur son développement, elle laisse à désirer, et il conseille d'utiliser une autre interface en lieu et place, à savoir Kibana. Notez que pour visualiser vos logs via une interface web, quelle que soit l'interface choisie des deux, vous devez absolument indexer vos messages dans une instance ElasticSearch (embarquée ou autonome).

6.2 Kibana

Kibana**[KIBANA]** est un frontend pour les données indexées dans ElasticSearch depuis Logstash. Projet écrit en PHP à l'origine, son auteur l'a porté en Ruby vu l'engouement des utilisateurs de Logstash. La combinaison Kibana/ElasticSearch est devenue l'association recommandée de Logstash pour visualiser les messages traités. Pour l'installer, c'est relativement rapide, comme indiqué dans la documentation officielle :

```
$ wget http://github.com/rashidkpc/Kibana/tarball/kibana-ruby -O /
tmp/kibana-ruby.tar.gz
$ sudo tar xzf /tmp/kibana-ruby.tar.gz -C /opt/
$ sudo mv /opt/rashidkpc-Kibana-* /opt/kibana/
$ sudo aptitude install -y ruby ruby-dev rubygems libcurl4-gnutls-dev
git build-essential
$ sudo gem install bundler
$ cd /opt/kibana/
$ sudo /var/lib/gems/1.8/bin/bundle  install
```

Au besoin, modifiez le fichier **KibanaConfig.rb** pour que le serveur écoute sur toutes les interfaces :

```
...
  # The adress ip Kibana should listen on. Comment out or set to
  # 0.0.0.0 to listen on all interfaces.
  #KibanaHost = '127.0.0.1'
  KibanaHost = '0.0.0.0'
...
```

En tant que tel, Kibana se démarre via Rack . Le plus flexible étant d'utiliser Passenger pour servir Kibana derrière un serveur web, comme Apache :

```
$ sudo aptitude install -y apache2 libapache2-mod-passenger
$ sudo chown  -R www-data /opt/kibana/
```

Modifiez le fichier */etc/apache2/sites-available/default* de l'hôte virtuel par défaut :

```
<VirtualHost *:80>
  ServerAdmin webmaster@localhost

        DocumentRoot /opt/kibana/public
        <Directory /opt/kibana/public>
         Options FollowSymLinks
         AllowOverride None
         Order allow,deny
         Allow from all
        </Directory>

  ErrorLog ${APACHE_LOG_DIR}/error.log
  LogLevel warn
  CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

```
$ sudo /etc/init.d/apache2 restart
```

Vous pouvez vous rendre avec votre navigateur sur le serveur Web, et commencer à visualiser vos logs. Vous pourrez d'ailleurs vous rendre compte que les informations extraites via Grok permettent de faire des recherches précises sur celles-ci (Figure 4 ci-contre).

Kibana est vraiment plus intuitive **[KBDEMO]** et facile à utiliser que l'interface incluse par défaut dans Logstash. Je vous renvoie vers la page et la vidéo de présentation du site officiel **[KBVIDEO]** pour découvrir ses fonctionnalités, dont celle de livestreaming, qui permet de voir s'afficher les logs traités en direct. De plus, le fait de l'avoir installé derrière Apache vous permet d'utiliser les différents mécanismes d'authentification du serveur, pour protéger votre interface.

6.3 Apache, Graphite et SatsD

N'oublions pas que, qui dit serveur web, dit nouveaux logs à traiter. Créez donc le fichier **/opt/logstash/conf/input-apache-file.log** suivant :

```
input {
 file {
   type => "apache"
   path => ["/var/log/apache2/*.log"]
 }
}
```

Les logs Apache sont différents de Syslog. Nous ne pouvons donc pas utiliser les mêmes filtres sur ces messages. Créez le fichier **/opt/logstash/conf/filter-apache.conf** suivant :

```
filter {
  grok {
    type => "apache"
    pattern => "%{COMBINEDAPACHELOG}"
  }

  date {
    type => "apache"
```

```
    match => [ "timestamp", "dd/MMM/yyyy:HH:mm:ss Z"]
    locale => "US"
  }
}
```

Nous désirons toujours parser ce message pour en extraire un maximum d'informations exploitables (agent, referrer, request, etc.). Le filtre grok possède déjà un motif **COMBINEDAPACHELOG**, pour nous aider en cela. Le format de date d'un log Apache est différent de celui de Syslog. Nous devons donc également utiliser un filtre **date** différent pour le type **apache** pour récupérer l'heure exacte du message. N'oubliez pas de redémarrer le service Logstash pour prendre en compte les modifications de configuration. Les messages du serveur Web devraient être traités correctement, avec des informations pertinentes extraites dans **@fields** (requête, réponse, nombre de bytes transférés) et la bonne date dans **@timestamp** :

```
{"@source"=>"file://linlogfabron/var/log/apache2/kibana.access.
log", "@tags"=>[], "@fields"=>{"clientip"=>["192.168.0.10"],
"ident"=>["-"], "auth"=>["-"], "timestamp"=>["04/
Mar/2013:23:06:52 +0100"], "verb"=>["GET"], "request"=>["/api/
search/eyJz=14"], "httpversion"=>["1.1"], "response"=>["200"],
"bytes"=>["4470"], "referrer"=>["\"http://linlogfabron/
index.html\""], "agent"=>["\"Mozilla/5.0 (X11; Linux x86_64)
AppleWebKit/537.1 (KHTML, like Gecko) Chrome/21.0.1180.81
Safari/537.1\""]}, "@timestamp"=>"2013-03-05T22:06:52.000Z",
"@source_host"=>"192.168.0.10", "@source_path"=>"/var/log/
apache2/kibana.access.log", "@message"=>"192.168.0.10 - - [04/
Mar/2013:23:06:52 +0100] \"GET /api/search/eyJz=14 HTTP/1.1\" 200
4470 \"http://linlogfabron/index.html\" \"Mozilla/5.0 (X11; Linux
x86_64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/21.0.1180.81
Safari/537.1\"", "@type"=>"apache"}
```

Et dans Kibana, les informations pertinentes extraites permettent à nouveau de faire des recherches précises dans vos logs (Figure 5).

Tant que nous y sommes, à chaque requête Apache et message généré, pourquoi ne pas alimenter un graphique dans Graphite ? En fait, on pourrait alimenter plusieurs graphiques

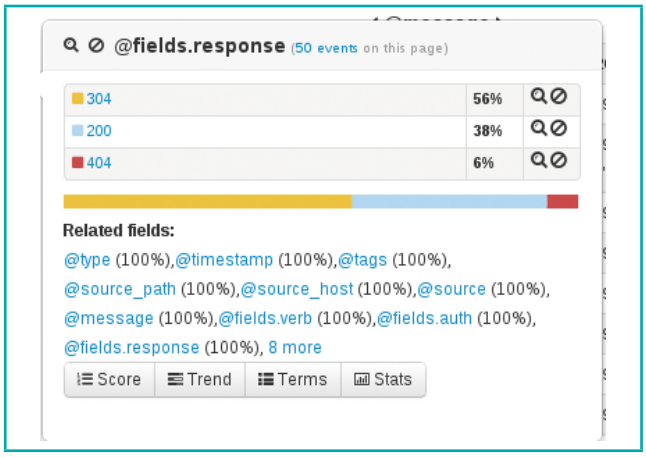


Figure 5

à partir d'un message : un sur le nombre de requêtes totales sur le serveur Web, un autre par type de réponse (200, 404, etc.), un graphique sur le nombre de bytes transféré, voire un graphique par localisation géographique des requêtes (avec le plugin **geoip** **[GEOIP]**). Par exemple, pour incrémenter des graphiques **apache.nom_du_serveur.response.type_de_réponse** par type de réponse (champ **@fields['response']**) et un autre graphique **apache.nom_du_serveur.bytes** sur la consommation réseau de ce serveur web (en utilisant le champ **@fields['bytes']**), on utiliserait le plugin d'output **statsd** à la configuration suivante :

```
output {
        statsd {
                type => "apache"
                host => "stastd-server"
                port => 8125
                increment => "apache.%{source}.response.%{response}"
                count => [ "apache.%{source}.bytes", "%{bytes}" ]
        }
}
```

Adieu Awstats !

Figure 4

Note

Dans la configuration précédente, on utilise le plugin `statsd` pour envoyer les métriques vers un serveur `stastd-server`, sur son port 8125. Pourquoi n'envoie-t-on pas directement vers un serveur Graphite ? Parce que les requêtes faites vers votre serveur web et les messages de logs correspondant générés, le seront de manière sporadique. Or je vous rappelle que Graphite nécessite de recevoir les messages à une certaine fréquence. Comme expliqué dans l'article sur Graphite, il faut donc passer par un service comme SatsD, qui va agréger les métriques, pour envoyer cette accumulation à fréquence régulière vers Graphite.

7 Clients

Vous avez désormais une installation serveur basique de Logstash, qui traite les logs locaux. Maintenant, comment envoyer les logs de vos différentes machines vers ce serveur ? Plusieurs possibilités s'offrent à vous.

Pour certaines machines, vous n'aurez pas tellement le choix. Appliance réseau, switch, serveurs Windows, certains équipements n'offrent comme possibilités de configuration le seul envoi des logs vers un serveur syslog. Pour ces équipements, notre input `tcp` à l'écoute sur le port 514 défini dans `/opt/logstash/conf/input-syslog-tcp.conf` est parfait. Croisez les doigts pour que les logs de vos matériels respectent le motif grok, ou sinon ajoutez le motif nécessaire au cas par cas.

Pour les serveurs sur lesquels vous avez une marge de manœuvre plus étendue, les possibilités sont forcément plus nombreuses. Le plus simple pour intégrer Logstash est encore d'utiliser Syslog, et de lui indiquer, en plus d'écrire dans les fichiers de logs habituels, d'envoyer les différents messages sur un port d'écoute du serveur Logstash, de manière similaire à ce qui précède. Notez qu'un serveur Syslog ne gère que les classiques `/var/log/syslog`, `/var/log/messages`, etc. Si ce serveur souhaite envoyer d'autres logs d'un service comme Apache, vous aurez à configurer Syslog pour lire et transférer les messages de ces fichiers de logs **[SYSLOGCONF]**.

Sinon, l'utilisation d'un client Logstash dédié est idéale pour envoyer vos logs vers un serveur Losgtash central. Dans ce cas, il existe différents clients Logstash pour faire office de Shipper **[CLIENTS]**. Je vous invite à utiliser Lumberjack **[LUMBERJACK]**, par le créateur de Logstash. Ce client va envoyer les logs choisis vers un serveur, qui devra utiliser le plugin d'entrée nommé `lumberjack`. Léger, son intérêt est de sécuriser les communications clients/serveurs, et il offre aussi un mécanisme de Broker type zmq/amqp : si le serveur est indisponible, le client met en tampon les messages, et retente leur envoi ultérieurement, vous évitant de perdre des logs.

Si vous pouvez vous le permettre en terme de ressources, vous pourrez même utiliser carrément le jar Logstash sur vos machines clientes, qui fonctionnera seulement en tant que Shipper. Si les machines clientes ne sont pas trop chargées, elles pourront même appliquer des filtres sur les messages, avant de les transférer, pour soulager le serveur central de ce travail. Sinon, vous transférez directement les messages vers un serveur Logstash central, qui s'occupera d'appliquer les filtres. Vous pourrez même utiliser le plugin d'output `lumberjack` de Logstash pour cela (utiliser le client Lumberjack est plus rapide pour l'instant, car les dernières optimisations apportées au client, n'ont pas été répercutées dans le code serveur de Logstash et de son plugin d'output `lumberjack`).

Sur le site de Logstash vous trouverez un exemple d'infrastructure répartie se basant sur Redis **[INFRAI]**. Sur le site de Kibana, une installation plus complexe est détaillée **[INFRA2]**, utilisant une queue RabbitMQ et des parseurs Logstash dédiés.

Je ne m'étendrai pas sur l'installation et la configuration des différents clients, que ce soit Syslog ou Lumberjack. Vous trouverez des exemples concrets et pratiques dans un extrait gratuit **[LOGSTASHBOOK]** du livre de James Turnbull **[KARTAR]**, contributeur de Logstash et Puppet, et administrateur système chevronné à qui l'on doit déjà de bons bouquins sur le sujet.

Conclusion

Il existe plusieurs autres solutions de Logging : Scribe, FluentD ou Graylog (vous pouvez d'ailleurs coupler ce dernier avec Logstash pour faire de la corrélation sur les événements**[GRAYLOG]**). Logstash est mature, il existe depuis 2008, et son créateur est maintenant employé par un gros hébergeur américain, pour travailler à temps plein sur son projet, ce qui assure sa pérennité. La version 1.1.10 est en approche (peut-être déjà sortie au moment où vous lirez cet article), avec tout un tas d'évolutions **[CHANGELOG]**. La volonté de l'auteur est d'améliorer les performances. Kibana évolue également, pour peut-être à terme être intégrée à Logstash. Le travail de son auteur a également plu aux développeurs d'ElasticSearch, qui l'ont intégré à leur équipe pour développer une interface web officielle à ElasticSearch **[KBDASHBOARD]**.

Cet article vous a aidé à mettre en place une infrastructure de logs, et vous devriez maintenant avoir une meilleure connaissance des événements sur vos serveurs,et pouvoir réagir plus rapidement en cas de problèmes. Dans cette continuité de mieux gérer votre infrastructure à l'aide d'outils

émergents, après le Graphing et le Logging, nous aborderons dans une troisième partie le monitoring avec deux nouveaux projets que sont Riemann et Sensu. ■

Références

[GLMF156] http://www.unixgarden.com/index.php/gnu-linux-magazine/gnulinux-magazine-n156-janvier-2012-en-kiosque/3

[10COMMANDEMENTS] http://www.masterzen.fr/2013/01/13/the-10-commandments-of-logging/

[LOGSTASH] http://logstash.net/

[ARGUMENTS] http://logstash.net/docs/latest/flags

[CONFIGURATION] http://logstash.net/docs/latest/configuration

[STDIN] http://logstash.net/docs/latest/inputs/stdin

[STDOUT] http://logstash.net/docs/latest/outputs/stdout

[PATTERNS] https://github.com/logstash/logstash/tree/master/patterns

[GROK-PATTERNS] https://github.com/logstash/logstash/blob/master/patterns/grok-patterns

[GROKDEBUG] http://grokdebug.herokuapp.com/

[JODA] http://joda-time.sourceforge.net/apidocs/org/joda/time/format/DateTimeFormat.html

[ESAPI] http://www.elasticsearch.org/guide/reference/api/

[GEM] https://github.com/jedi4ever/logstash-cli

[WIKI] https://github.com/logstash/logstash/wiki/Elasticsearch-Storage-Optimization

[ESTPL] http://www.elasticsearch.org/guide/reference/api/admin-indices-templates.html

[ESCLEAN] https://logstash.jira.com/browse/LOGSTASH-211

[LOGSTASH.INIT] http://sebsauvage.net/paste/?e62fbcfa62ee4077#3uWl6wJ8GEoAkK3Zb06sPHgTmBJHj2Rp8pesCHHY3Ro=

[KIBANA] http://kibana.org/

[KBDEMO] http://demo.logstash.net

[KBVIDEO] http://kibana.org/about.html

[GEOIP] http://logstash.net/docs/latest/filters/geoip

[SYSLOGCONF] http://cookbook.logstash.net/recipes/rsyslog-agent/

[CLIENTS] http://cookbook.logstash.net/recipes/log-shippers/

[LUMBERJACK] https://github.com/jordansissel/lumberjack

[INFRA1] http://logstash.net/docs/latest/tutorials/getting-started-centralized

[INFRA2] http://kibana.org/infrastructure.html

[LOGSTASHBOOK] http://www.logstashbook.com/TheLogStashBook_sample.pdf

[KARTAR] http://www.kartar.net/

[GRAYLOG] http://wiki.monitoring-fr.org/infra/logstash#configuration-graylog2

[CHANGELOG] https://github.com/logstash/logstash/blob/master/CHANGELOG

[KBDASHBOARD] https://github.com/elasticsearch/kibana-dashboard

HAUTE DISPONIBILITÉ ET RÉPARTITION DE CHARGE AVEC KEEPALIVED !

par Julien Morot
[Passionné de logiciels libres depuis 1998]

Keepalived c'est un peu comme le miel, au début ça colle aux doigts et après on s'aperçoit que son parfum sucré est drôlement agréable. Je vais donc aborder les fondements de la haute disponibilité de services réseaux au niveau kernel pour arriver à une solution entièrement pilotée.

1 De quoi s'agit-il ?

Une architecture informatique est dite hautement disponible lorsque sa conception est pensée d'une part pour répondre à la panne d'un ou plusieurs équipements et d'autre part lorsqu'elle est capable d'encaisser un pic de charge sans sourciller. La solution présentée dans ces pages se focalise sur la notion de services réseaux. Bien entendu, la notion de haute disponibilité doit être prise en compte à tous les niveaux d'un système, donc aussi bien au niveau matériel (RAID, liens réseaux redondés...) qu'au niveau applicatif (clusters MySQL, LDAP multi-master, etc.) qu'humain (hé oui si l'unique sysadmin qui connaît une technologie est en vacances lors d'une panne...). La répartition de charge consiste à pouvoir distribuer les requêtes de manière efficace sur un pool de serveurs afin de pouvoir gérer un nombre important de clients.

Un mode de fonctionnement très simple pour faire de la haute disponibilité consiste à mettre en place ce que l'on appelle du Round Robin DNS, en faisant pointer plusieurs enregistrements DNS identiques, par exemple :

```
www IN  A 192.168.1.61
www IN  A 192.168.1.62
```

Cependant, cette solution ne tient pas compte de la disponibilité réelle des services et peut connaître des problèmes de cache DNS. Il convient donc pour réaliser un fonctionnement propre d'utiliser un load balancer en frontal. Dans cette topologie, l'ensemble des flux entrants passent par le répartiteur qui achemine aux nœuds le trafic selon un algorithme (round robin, round robin pondéré, hash IP source, moins de connexions, etc.).

2 Comment ça fonctionne sous Linux ?

2.1 La couche IPVS

IPVS pour IP Virtual Server est une interface qui se greffe au niveau de la couche Netfilter. Pour rappel, Netfilter et la couche du noyau en charge de la manipulation des paquets réseaux. C'est cette couche que vous manipulez lorsque vous utilisez la commande **`iptables`** afin de mettre en place des règles de firewall. C'est cette même couche qui sera manipulée par la commande **`ipvsadm`** présente suite à l'installation du package éponyme.

L'opération consiste donc à présenter aux clients une adresse IP dite virtuelle (VIP) indépendante de celle utilisée pour l'administration des machines qui redirigera les flux vers les serveurs réels.

IPVS est capable d'opérer selon trois modes :

- IPVS-TUN, c'est-à-dire derrière un tunnel de type encapsulation IP. Je n'en parlerai pas, car à ma connaissance ce mode est extrêmement peu utilisé ;
- IPVS-NAT ;
- IPVS-DR.

2.2 IPVS-NAT

Le mode IPVS-NAT est le plus simple à mettre en œuvre techniquement, mais implique une plus grande maîtrise du réseau. Il permet également d'utiliser un adressage IPV4 privé pour les serveurs réels lorsque l'on ne dispose pas de suffisamment d'adresses IP publiques (voir Figure 1).

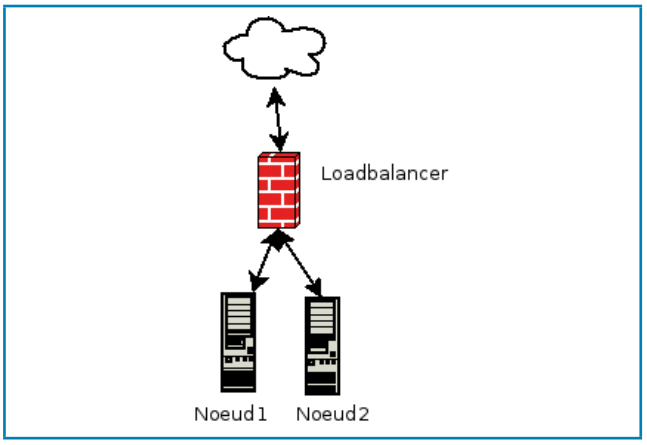


Figure 1

Dans ce mode, lorsqu'un paquet arrive sur le load balancer et qu'il correspond à un service fourni par IPVS, un serveur réel est choisi dans le pool selon l'algorithme sélectionné. Un enregistrement dans la table des connexions est effectué pour préserver la destination du paquet. L'adresse IP est réécrite pour correspondre à celle du serveur réel puis le paquet est routé vers celui-ci. Lorsque le serveur réel répond, le load balancer masque son IP en réécrivant le paquet puis l'achemine au client. Ainsi, tous les flux entrants et sortants passent par le load balancer.

2.3 IPVS-DR

Dans ce mode de fonctionnement, le load balancer et les serveurs réels sont physiquement sur le même segment de réseau. La VIP est partagée entre le load balancer et les serveurs réels via une pseudo-interface. Lorsqu'une requête arrive sur la VIP, le load balancer la dispatche sur les serveurs réels en fonction de l'algorithme retenu.

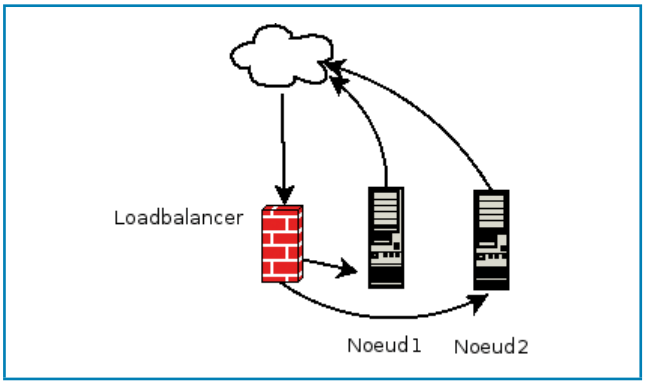


Figure 2

Sur le serveur de destination, la requête est traitée puis la réponse est directement effectuée par le serveur réel au client, sans passer cette fois par le load balancer. La décision de routage effectuée par le load balancer est conservée dans une table afin de conserver le routage jusqu'au timeout ou jusqu'à la terminaison de la connexion. Ce mode de fonctionnement a l'avantage d'être plus performant comme les réponses ne passent pas par le load balancer.

3 Mise en bouche avec IPVS

Commençons la pratique en se basant sur une architecture réseau simple. L'infrastructure de test est constituée de deux noeuds **srv1** et **srv2** d'adresses IP 192.168.1.61 et .192.168.1.62. Les deux nœuds hébergent chacun un serveur web Apache. En frontal, nous aurons un load balancer **lb** d'adresse IP 192.168.1.50. En ce qui concerne la distribution, elle a assez peu d'importance même si rien ne vaut une Debian !

Après avoir vidé les éventuels enregistrements ipvs déjà présents, on ajoute un service réseau sur le port 80 correspondant donc au HTTP, associé à un algorithme d'équilibrage. J'utilise ici les options longues afin de rendre les commandes plus explicites, mais des options courtes existent bien entendu.

```
root@lb:~# ipvsadm --clear
root@lb:~# ipvsadm --add-service --tcp-service 192.168.1.50:80 --scheduler wrr
```

On peut vérifier que le service réseau est pris en compte en saisissant simplement le nom de la commande :

```
root@lb:~# ipvsadm
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  192.168.1.60:http rr
```

On ajoute ensuite les deux nœuds sur le load balancer :

```
root@lb:~# ipvsadm --add-server --tcp-service 192.168.1.50:80 --real-
server 192.168.1.61:80 --gatewaying --weight 100
root@lb:~# ipvsadm --add-server --tcp-service 192.168.1.50:80 --real-
server 192.168.1.62:80 --gatewaying --weight 100
```

Il est désormais possible de faire pointer un navigateur sur la VIP (donc http://192.168.1.50) et constater qu'il ne se passe... absolument rien !

Et pourtant, le load balancer retransmet bien des paquets :

```
root@lb:~# ipvsadm -L -n --stats
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port               Conns   InPkts  OutPkts  InBytes OutBytes
  -> RemoteAddress:Port
TCP  192.168.1.50:80                     6       42        0     2520        0
  -> 192.168.1.61:80                     3       21        0     1260        0
```

Une inspection de paquets nous le confirme :

```
root@srv1:~# tcpdump port 80
20:23:56.204722 IP new-host-2.home.46973 > 192.168.1.50.http: Flags
[S], seq 95994012, win 14600, options [mss 1460,sackOK,TS val 697670
ecr 0,nop,wscale 7], length 0
20:23:56.457979 IP new-host-2.home.46975 > 192.168.1.50.http: Flags
[S], seq 3006976356, win 14600, options [mss 1460,sackOK,TS val
697746 ecr 0,nop,wscale 7], length 0
```

En effet, les nœuds du cluster reçoivent des paquets venant de l'adresse IP du load balancer sur l'adresse IP virtuelle qui ne correspond donc pas à une interface d'écoute. La solution consiste à faire de l'IP aliasing sur l'IP du load balancer. Cependant, le risque est que les nœuds parasitent les caches ARP en répondant aux requêtes. La solution consiste à ignorer les requêtes ARP. Comme je suis plutôt ceinture et bretelles, je préfère également bloquer les requêtes à coup d'arptables au montage de l'interface.

Sur les deux nœuds de notre cluster, on va donc ajouter ceci au fichier **/etc/sysctl.conf** :

```
net.ipv4.ip_forward=0
net.ipv4.conf.all.arp_ignore=1
net.ipv4.conf.all.arp_announce=2
net.ipv4.conf.default.arp_ignore=1
net.ipv4.conf.default.arp_announce=2
net.ipv4.conf.lo.arp_ignore=1
net.ipv4.conf.lo.arp_announce=2
```

Et paramétrer l'interface virtuelle comme ceci dans le fichier **/etc/netwowk/interfaces** (ou pour une RedHat/CentOS, paramétrer un fichier **/etc/sysconfig/network-scripts/ifcfg-lo.0** adapté) :

```
auto lo:0
iface lo:0 inet static
        address 192.168.1.60
        netmask 255.255.255.255
        up arptables -A INPUT -d 192.168.1.60 -j DROP
```

On active le tout :

```
sysctl -p /etc/sysctl.conf  && ifup lo:0
```

Si vous ouvrez votre navigateur sur l'URL que l'on a indiqué plus haut, ça doit fonctionner !

4 Et si on faisait de la vraie haute disponibilité cette fois ?

4.1 Les défauts de l'architecture précédente

Ce que l'on a mis en place est fonctionnel, mais il faut reconnaître que c'est très loin d'être parfait. D'une part, on parle de haute disponibilité alors que l'on a introduit une machine : le load balancer qui n'est absolument pas redondé. D'autre part, notre configuration ne tient absolument pas compte de l'état réel des nœuds. Pour autant, les interruptions de service autant normales (maintenances programmées ou non…) qu'anormales (pannes…) font partie de la vie d'un serveur.

C'est à ce niveau que le logiciel Keepalived intervient. Il implémente le protocole VRRP (Virtual Router Redundancy Protocol). Avec ce protocole, les deux équipements participant à un groupe VRRP communiquent en multicast afin de définir un rôle maître-esclave dans le but de s'échanger une adresse IP virtuelle en cas de défaillance d'un des deux équipements. D'autre part, Keepalived possède la notion de « checks » afin de vérifier qu'un service réseau est accessible et ainsi dynamiser la configuration de la couche **ipvsadm**.

4.1 Mise en place

Pour cela, on va conserver nos nœuds existants, mais partir de deux nouveaux load balancer **lb1** et **lb2** d'adresses IP 192.168.1.51 et 192.168.1.52. Sur chaque nœud, on va installer le package **keepalived** et construire un fichier **/etc/keepalived/keepalived.conf**, qui sera à quelques détails près identique sur les deux nœuds du cluster.

La première section, **global_defs**, comporte principalement des informations concernant les notifications par mail. La notion de **routeur_id** a somme toute peu d'importance, car elle constitue un label qui peut être défini sur le hostname du serveur :

```
global_defs {
   # Configuration des notifications :
   notification_email {
     sysadmin@domain.local
     root@domain.local
   }
   notification_email_from nagios@domain.local
   smtp_server 192.168.1.16
   smtp_connect_timeout 30

   # ID du load balancer
   router_id lb1
}
```

Ensuite, il faut configurer le groupe VRRP. Celui-ci se définit via un identifiant VRRP, le **virtual_routeur_id** qui est commun aux deux load balancer. Ensuite, on indique un état par défaut du load balancer, le paramètre **state** positionné à **MASTER** sur lb1 et **BACKUP** sur lb2. Cependant, c'est la priorité la plus haute qui a réellement le dernier mot. Enfin, on définit la VIP qui sera partagée entre les deux load balancer.

```
vrrp_instance VI_1 {
   virtual_router_id 100
   state MASTER
   priority 100
   # Check inter-load balancer toutes les 1 secondes
   advert_int 1
   # Synchro de l'état des connexions entre les LB sur l'interface
eth0
   lvs_sync_daemon_interface eth0
   interface eth0
   # Authentification mutuelle entre les LB, identique sur les deux
membres
```

```
   authentication {
        auth_type PASS
        auth_pass secret
   }
   # Interface réseau commune aux deux LB
   virtual_ipaddress {
        192.168.1.50/32 brd 192.168.1.255 scope global
   }
}
```

Enfin, la dernière partie intègre l'algorithme d'équilibrage de charge ainsi que les serveurs réels. On retrouve donc une configuration extrêmement proche de ce que nous avons vu en étudiant IPVS ce qui rend au final la lecture de la configuration relativement limpide.

```
virtual_server 192.168.1.50 80 {
    # L'etat de santé des cibles est vérifié toutes les 5 secondes
    delay_loop 5
    lb_algo wrr
    lb_kind DR
    # Seul le protocole TCP est implémenté.
    protocol TCP
    # Si la VIP n'a pas pu être activée, le
    # contrôle des serveurs réels est suspendu.
    ha_suspend
    # Vous pouvez définir l'adresse d'un serveur vers qui rediriger les
    # requêtes si tous les serveurs réels sont injoignables.
    #sorry_server 123.45.67.12 80
    real_server 192.168.1.61 80 {
        weight 1

        HTTP_GET {
            url {
                path /
                digest 21dde95d9d269cbb2fa6560309dca40c
            }
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
        }
    }

        real_server 192.168.1.62 80 {
        weight 1

        HTTP_GET {
            url {
              path /
              digest 21dde95d9d269cbb2fa6560309dca40c
            }
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
        }
    }
}
```

Un point particulier concerne cependant le digest. Il s'agit d'un hash de la page à vérifier. Bien entendu, les données des serveurs doivent être identiques pour retourner le même hash. Celui-ci s'obtient avec une commande fournie par le paquet keepalived et s'exécute de cette façon en direction d'un des nœuds :

```
root@lb1:/etc/keepalived#  genhash -s 192.168.1.62 -p 80 -u /
MD5SUM = 21dde95d9d269cbb2fa6560309dca40c
```

Keepalived possède une sonde dédiée à la gestion du flux HTTP, mais il aurait tout à fait été possible d'utiliser le check générique **TCP_CHECK** comme ci-après. Bien entendu, avec ce type de check, on ne vérifie que la disponibilité d'un port, mais absolument pas la restitution correcte des informations.

```
real_server 192.168.1.61 80 {
    weight 1
    TCP_CHECK {
        connect_timeout 3
        connect_port 80
    }
}
```

4.2 Ça marche vraiment ?

Maintenant que tout est en ordre et la configuration déployée sur les deux nœuds, il ne reste plus qu'à démarrer keepalived via le script d'init ou bien via la commande **keepalived -f /etc/keepalived/keepalived.conf**.

Ceux qui auront saisi la commande **ifconfig** sur le master après ça auront peut-être eu une interrogation : comment mon

load balancer peut rediriger les flux pour la VIP 192.168.1.60 alors qu'elle n'apparaît pas dans la sortie d'**ifconfig** ? C'est simple, il faut passer par **iproute2** qui prend réellement en charge toutes les fonctionnalités du noyau :

```
root@lb1:~# ip addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP qlen 1000
    link/ether 52:54:00:42:b0:b3 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.51/24 brd 192.168.1.255 scope global eth0
    inet 192.168.1.50/32 brd 192.168.1.255 scope global eth0
    inet6 fe80::5054:ff:fe42:b0b3/64 scope link
       valid_lft forever preferred_lft forever
```

Sur le master, les logs doivent indiquer que les checks sont activés et qu'il est bien passé en mode master :

```
May 25 23:29:57 lb1 Keepalived_healthcheckers: Activating
healtchecker for service [192.168.1.61]:80
May 25 23:29:57 lb1 Keepalived_healthcheckers: Activating
healtchecker for service [192.168.1.62]:80
May 25 23:29:57 lb1 kernel: [  417.914963] IPVS: sync thread started:
state = MASTER, mcast_ifn = eth0, syncid = 100
May 25 23:29:57 lb1 Keepalived_vrrp: VRRP_Instance(VI_1) Transition
to MASTER STATE
May 25 23:29:58 lb1 Keepalived_vrrp: VRRP_Instance(VI_1) Entering
MASTER STATE
```

Même chose sur le load balancer secondaire, mais en indiquant cette fois le passage en mode secours :

```
May 25 23:30:45 lb2 Keepalived_vrrp: VRRP_Instance(VI_1) Entering BACKUP STATE
May 25 23:30:45 lb2 Keepalived_healthcheckers: Using LinkWatch kernel netlink
reflector...
May 25 23:30:45 lb2 Keepalived_healthcheckers: Activating healtchecker for
service [192.168.1.61]:80
May 25 23:30:45 lb2 Keepalived_healthcheckers: Activating healtchecker for
service [192.168.1.62]:80
May 25 23:30:45 lb2 kernel: [  476.743238] IPVS: sync thread started: state =
BACKUP, mcast_ifn = eth0, syncid = 100
```

Simulons une panne ou une maintenance d'un des serveurs en arrêtant **srv1**.

```
May 25 23:32:57 lb2 Keepalived_healthcheckers: Timeout connect,
timeout server [192.168.1.61]:80.
May 25 23:32:57 lb2 Keepalived_healthcheckers: Removing service
[192.168.1.61]:80 from VS [192.168.1.50]:80
```

Puis son retour à la normale :

```
May 25 23:34:10 lb1 Keepalived_healthcheckers: MD5 digest success to
[192.168.1.61]:80 url(1).
May 25 23:34:15 lb1 Keepalived_healthcheckers: Remote Web server
[192.168.1.61]:80 succeed on service.
```

Même chose, en simulant une panne du load balancer maître, les logs du secondaire indiquent qu'il prend le relais. La commande **ip addr** doit confirmer la bascule de la VIP.

```
May 25 23:35:21 lb2 Keepalived_vrrp: VRRP_Instance(VI_1) Transition
to MASTER STATE
May 25 23:35:22 lb2 Keepalived_vrrp: VRRP_Instance(VI_1) Entering
MASTER STATE
May 25 23:35:22 lb2 kernel: [  753.871114] IPVS: stopping backup sync
thread 1959 ...
May 25 23:35:22 lb2 kernel: [  753.873973] IPVS: sync thread started:
state = MASTER, mcast_ifn = eth0, syncid = 100
```

À noter, la directive **nopreempt** permet d'empêcher la bascule vers le maître par défaut en cas de retour à la normale de celui-ci.

4.3 J'ai comme un doute sur ta manière de vérifier là...

Comme je le disais un peu plus haut, en dehors du check HTTP, les checks TCP sont extrêmement simples et ne vérifient que la possibilité de se connecter sur un port donné. Keepalived fournit un check générique, **MISC_CHECK**, permettant de déléguer à un script externe la vérification de l'état de santé. C'est le code de retour du script qui indique l'état de santé. Un code 0 renvoie un serveur disponible, 1 une erreur. D'autres valeurs sont possibles et sont documentées dans la page de man de **keepalived.conf**.

Un exemple pourrait donc être pour un serveur LDAP :

```
#!/bin/bash
HOST=$1
BASEDN="dc=domain,dc=local"
ldapsearch -x -h $HOST -b $BASEDN > /dev/null
```

Qui serait appelé dans keepalived comme suit :

```
real_server 192.168.1.61 389 {
    weight 1
    MISC_CHECK {
        misc_path "/usr/local/bin/check_ldap.sh 192.168.1.61"
    }
}
real_server 192.168.1.62 389 {
    weight 1
    MISC_CHECK {
        misc_path "/usr/local/bin/check_ldap.sh 192.168.1.61"
    }
}
```

Conclusion

Voilà, je pense que l'on a un aperçu de ce qu'il est possible de faire grâce aux logiciels libres et de façon relativement simple sans devoir passer à la caisse pour acquérir une appliance « boite noire ». D'autres logiciels à la manière de keepalived sont capables de piloter la couche IPVS comme **ldirectord**, piranha qui offrent des fonctionnalités similaires et peuvent mériter votre attention, mais ils se basent tous sur les mêmes principes ! ■

Abonnez-vous !

Profitez de nos offres d'abonnement spéciales disponibles au verso !

Téléphonez au 03 67 10 00 20 ou commandez par le Web

Économisez plus de 25%*

* Sur le prix de vente unitaire France Métropolitaine

11 Numéros de GNU/Linux Magazine

Les 3 bonnes raisons de vous abonner :

- Ne manquez plus aucun numéro.
- Recevez GNU/Linux Magazine chaque mois chez vous ou dans votre entreprise.
- Économisez 24,50 €/an ! (soit plus de 3 magazines offerts !)

4 façons de commander facilement :

- par courrier postal en nous renvoyant le bon ci-dessous
- par le Web, sur boutique.ed-diamond.com
- par téléphone, entre 9h-12h et 14h-18h au 03 67 10 00 20
- par fax au 03 67 10 00 21

ABONNEMENT

11 GNU LINUX MAGAZINE / FRANCE

58€*

au lieu de 82,50 €* en kiosque

Économie : 24,50 €*

*OFFRE VALABLE UNIQUEMENT EN FRANCE MÉTROPOLITAINE
Pour les tarifs hors France Métropolitaine, consultez notre site : boutique.ed-diamond.com

Bon d'abonnement à découper et à renvoyer à l'adresse ci-dessous

Tournez SVP pour découvrir toutes les offres d'abonnement >>

Voici mes coordonnées postales :	
Société :	
Nom :	
Prénom :	
Adresse :	
Code Postal :	
Ville :	
Pays :	
Téléphone :	
e-mail :	

☐ Je souhaite recevoir les offres promotionnelles et newsletter des Éditions Diamond.

☐ Je souhaite recevoir les offres promotionnelles de nos partenaires.

GNU LINUX MAGAZINE / FRANCE

Édité par Les Éditions Diamond
Service des Abonnements
B.P. 20142 - 67603 Sélestat Cedex
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : **boutique.ed-diamond.com/cgv** et reconnais que ces conditions de vente me sont opposables.

Tournez SVP pour découvrir toutes les offres d'abonnement

PROFITEZ DE NOS OFFRES D'ABONNEMENT SPÉCIALES POUR LIRE PLUS ET FAIRE DES ÉCONOMIES !

➡ Abonnement

offre 1 — ABONNEMENT — 11 LINUX MAGAZINE / FRANCE — **58€*** — au lieu de 82,50€** en kiosque — Economie : 24,50 €

Vous pouvez également vous abonner sur :

boutique.ed-diamond.com

ou par Tél. : +33 (0)3 67 10 00 20 /
Fax : +33 (0)3 67 10 00 21

NOUVEAU !

numerique.ed-diamond.com
pour vous abonner et acheter vos magazines en format numérique (PDF)

unixgarden.com
pour retrouver une sélection d'articles des Éditions Diamond

* Tarifs France Métro (F)
** Base tarifs kiosque zone France Métro (F)

➡ Voici nos offres d'abonnements groupés incluant GLMF

offre 3 — ABONNEMENTS GROUPÉS — 11 LINUX MAGAZINE / FRANCE + 6 LINUX PRATIQUE — **85€*** — au lieu de 121,50€** en kiosque — Economie : 36,50 €

offre 4 — ABONNEMENTS GROUPÉS — 11 LINUX MAGAZINE / FRANCE + 6 Hors-séries — **89€*** — au lieu de 130,50€** en kiosque — Economie : 41,50 €

offre 5 — ABONNEMENTS GROUPÉS — 11 LINUX MAGAZINE / FRANCE + 6 MISC — **90€*** — au lieu de 133,50€** en kiosque — Economie : 43,50 €

offre 6 — ABONNEMENTS GROUPÉS — 11 LINUX MAGAZINE / FRANCE +6 HS + 6 LINUX PRATIQUE — **119€*** — au lieu de 169,50€** en kiosque — Economie : 50,50 €

offre 7 — ABONNEMENTS GROUPÉS — 11 LINUX MAGAZINE / FRANCE +6 HS + 6 MISC — **124€*** — au lieu de 181,50€** en kiosque — Economie : 57,50 €

offre 8 — ABONNEMENTS GROUPÉS — 11 LINUX MAGAZINE / FRANCE +6 HS + 6 LINUX PRATIQUE + 6 MISC — **154€*** — au lieu de 220,50€** en kiosque — Economie : 66,50 €

offre 9 — ABONNEMENTS GROUPÉS — 11 LINUX MAGAZINE / FRANCE +6 HS + 6 LINUX PRATIQUE + 6 MISC + 6 LINUX ESSENTIEL — **184€*** — au lieu de 259,50€** en kiosque — Economie : 75,50 €

offre 12 — ABONNEMENTS GROUPÉS — 11 LINUX MAGAZINE / FRANCE +6 HS + 6 LINUX PRATIQUE +3 HS + 6 MISC +2 HS + 6 LINUX ESSENTIEL — **215€*** — au lieu de 301,50€** en kiosque — Economie : 86,50 €

offre 2 — ABONNEMENTS GROUPÉS — 6 LINUX ESSENTIEL + 6 LINUX PRATIQUE — **60€*** — au lieu de 78,00€** en kiosque — Economie : 18,00 €

➡ Voici nos autres offres d'abonnements groupés

offre 10 — ABONNEMENTS GROUPÉS — 6 MISC + 2 Hors-séries — **48€*** — au lieu de 69,00€** en kiosque — Economie : 21,00 €

offre 11 — ABONNEMENTS GROUPÉS — 6 LINUX PRATIQUE + 3 Hors-séries — **48€*** — au lieu de 63,00€** en kiosque — Economie : 15,00 €

offre 15 — ABONNEMENTS GROUPÉS — 6 LINUX ESSENTIEL + 6 LINUX PRATIQUE +3 HS — **78€*** — au lieu de 102,00€** en kiosque — Economie : 24,00 €

➡ Nos Tarifs s'entendent TTC et en euros

		F France Métro	D DOM	T TOM	Zone 1 Europe	Zone 2 Afrique / Orient	Zone 3 Amérique	Zone 4 Asie / Océanie
1	Abonnement GLMF	58 €	78 €	94 €	80 €	87 €	84	80 €
2	Abonnement LPE + LP	60 €	86 €	105 €	88 €	96 €	92 €	89 €
3	Abonnement GLMF + LP	85 €	120 €	145 €	123 €	134 €	129 €	124 €
4	Abonnement GLMF + GLMF HS	89 €	122 €	147 €	125 €	136 €	131 €	126 €
5	Abonnement GLMF + MISC	90 €	128 €	151 €	130 €	141 €	136 €	131 €
6	Abonnement GLMF + GLMF HS + Linux Pratique	119 €	164 €	198 €	168 €	183 €	176 €	170 €
7	Abonnement GLMF + GLMF HS + MISC	124 €	172 €	204 €	175 €	190 €	183 €	177 €
8	Abonnement GLMF + GLMF HS + MISC + LP	154 €	214 €	255 €	218 €	237 €	228 €	221 €
9	Abonnement GLMF + GLMF HS + MISC + LP + LPE	184 €	258 €	309 €	263 €	286 €	275 €	266 €
10	Abonnement MISC + MISC HS	48 €	66 €	76 €	66 €	72 €	69 €	68 €
11	Abonnement LP + LP HS	48 €	65 €	78 €	66 €	72 €	70 €	68 €
12	Abonnement GLMF + GLMF HS + MISC + MISC HS + LP + LP HS + LPE	215 €	297 €	355 €	302 €	329 €	317 €	307 €
15	Abonnement LPE + LP + LP HS	78 €	109 €	132 €	111 €	121 €	117 €	113 €

- ZONE 1 : Allemagne, Belgique, Danemark, Italie, Luxembourg, Norvège, Pays-Bas, Portugal, Suède, Autriche, Espagne, Finlande, Grande Bretagne, Grèce, Islande, Suisse, Irlande, Estonie, Croatie, Slovenie, Slovaquie, Republique Tcheque, Pologne, Bielorussie, Bosnie Herzegovine, Bulgarie, Chypre, Georgie, Hongrie, Lettonie, Lituanie, Macedoine, Malte, Moldova, Roumanie, Russie, Serbie, Ukraine, Albanie, Armenie, ...
- ZONE 2 : Algerie, Maroc, Tunisie, Turquie, Afrique du Sud, Seychelle, Senegal, Israel, Palestine, Syrie, Jordanie, Botswana, Cameroun, Cap Vert, Comores, Rep.Dom. Congo, Cote d'Ivoire, Egypte, Kenya, Libye, Madagascar, Nigeria, ...
- ZONE 3 : Canada, Etats Unis, Guyana, Haiti, Republique Dominicaine, Jamaique, Argentine, Bresil, Cuba, Mexique, ...
- ZONE 4 : Australie, Japon, Chine, Coree du Nord, Coree du Sud, Inde, Indonesie, Nouvelle Zelande, Taiwan, Thailande, Vietnam, ...

Mes choix :

Mon 1er choix	Je sélectionne le N° (1 à 15) de l'offre choisie :	
Mon 2ème choix	Je sélectionne le N° (1 à 15) de l'offre choisie :	
Mon 3ème choix	Je sélectionne le N° (1 à 15) de l'offre choisie :	
	Je sélectionne ma zone géographique (F à Zone 4) :	
	J'indique la somme due : (Total)	€

Exemple : je souhaite m'abonner à l'offre GNU/Linux Magazine + GNU/Linux Magazine Hors-série + MISC (offre 7) et je vis en Belgique (zone 1), ma référence est donc 7zone1 et le montant de l'abonnement est de 175 euros.

Je choisis de régler par :

- ❑ Chèque bancaire ou postal à l'ordre des Éditions Diamond
- ❑ Carte bancaire n° |_|_|_|_| |_|_|_|_| |_|_|_|_| |_|_|_|_|
  Expire le : |_|_|_|_|
  Cryptogramme visuel : |_|_|_|

Date et signature obligatoire

JAVA EST-IL UN LANGAGE OBJET ?

par Fabienne Sandoz

Java est un langage qui résulte d'une construction entre le monde du langage C (ou C++) et les langages objets tels que Smalltalk. Il a essayé de prendre le meilleur des deux mondes, c'est-à-dire la portabilité et la facilité de maintenance des langages objets tout en gardant un niveau de performance acceptable. Des contrôles (notamment de typages) sont apportés par le compilateur.

1 Qu'à pris Java de chacun des deux mondes ?

Du côté objet :

- La possibilité de faire des classes, de les instancier ;
- La présence d'une machine virtuelle pour la portabilité ;
- Le garbage collector pour une utilisation simplifiée de la mémoire.

De l'autre :

- Les types primitifs pour des opérations mathématiques plus performantes ;
- Le typage fort pour des contrôles à la compilation et une meilleure performance ;
- La syntaxe et la structure du langage (boucle, test).

Ce langage résulte donc d'un compromis, mais on peut se demander s'il ne s'agit pas de l'alliance de la carpe et du lapin.

En étudiant différents exemples, on peut mieux se rendre compte des manques qu'il y a dans le monde java par rapport à un langage « pur objet ». Ces manques ont été compensés de façon plus ou moins satisfaisante.

2 Premier exemple : les Factory

Aujourd'hui, une pratique considérée comme bonne et préconisée par la communauté java est l'utilisation généralisée des Factory.

Ainsi, au lieu de faire :

```
Personne p = new Personne() ;
```

Je dois faire créer une classe **PersonneFactory** :

```
public class PersonneFactory {

  public static Personne newPersonne() {
    return new Personne();
  }
}
```

Et pourquoi dois-je faire tout ça ? Je crée une classe juste pour instancier des objets ? À quoi ça sert ?

On sait depuis le début des langages objets que l'instanciation est coûteuse. Créer des objets prend de la place mémoire. (Mais comment fait-on de l'objet sans créer des objets ???).

Avec les factory, on se laisse la possibilité de redéfinir l'instanciation, pour passer par exemple par un pool d'objets, ou alors en travaillant par clonage d'objets. Si les **new** sont disséminés partout dans le code, on doit tout remodifier. Alors qu'avec une Factory on ne change qu'une seule méthode.

Donc pour éventuellement me laisser la possibilité d'évoluer demain, je multiplie les classes et je remplace une ligne de code (**new Personne()**) par une autre (**PersonneFactory.newPersonne()**).

Pourquoi est-ce un artifice ?

Et bien en smalltalk par exemple, je n'ai pas besoin de factory. Si je veux changer l'instanciation, je peux redéfinir **new** qui est une méthode.

new est une méthode qui s'applique à la métaclasse (c'est-à-dire la classe de la classe).

Demain, je pourrai redéfinir la méthode **new** de la classe **Personne class** pour utiliser un pool, recycler des instances, ou autre.

Ceci n'est pas possible en java, car les classes ne sont pas objets. Elles n'ont pas de métaclasse. **new** n'est pas une méthode. C'est une instruction du langage. Les Factory sont un contournement pour pallier à cette absence.

3 Deuxième exemple : Les interfaces

Dans une classe Console, j'écris la méthode suivante :

```
public static void afficher(Personne p) {
  System.out.println("NOM : " + p.getNom());
  System.out.println("né le :"
+p.getDateDeNaissance());
//(pour simplifier j'ai oublié le formattage
de la date...)
}
```

Ensuite, je me rends compte que mon application gère aussi des chats et des chiens qui ont aussi un nom et une date de naissance. Mais je ne peux pas appeler **Console.afficher(chat)**. Le compilateur me l'interdit, car chat n'est pas une personne.

Pour afficher des chats et des chiens avec ma méthode, je dois :

- faire hériter chat et chien de personne (pas terrible)
- ou créer une interface **Affichable** :
  - → **Affichable** contient les deux signatures
  - → chat et chien implémentent **Affichable**
  - → je modifie Personne pour implémenter **Affichable**
  - → je modifie la méthode **afficher** pour qu'elle prenne un paramètre de type **Affichable**

Je dois donc modifier mon code pour le faire évoluer. Les interfaces sont une solution apportée à un problème que pose le typage.

En Smalltalk (langage non typé), je peux appeler **afficher** sur n'importe quelle instance, sans aucune modification. Le risque est que cela plante à l'exécution.

```
afficher: unTruc
  Transcript show: NOM : " ;
    show:  unTruc nom;
    show: né le : ".
    show: unTruc dateDeNaissance;
    cr
```

```
| personne  chat chien voiture |
personne:= Personne new.
chat := Chat new.
chien := Chien new.
voiture := Voiture new.
[...]
Console afficher: personne.
Console afficher: chat.
Console afficher: chien.
Console afficher: voiture.
```

Si voiture n'a pas de méthode **nom** et/ou **dateDeNaissance**, je vais obtenir un debugguer avec le message **voiture doesNotUnderstand : #nom**.

En principe, l'objet sert à faire du polymorphisme (le comportement d'une instance est différent suivant la classe à laquelle elle appartient). Donc le comportement doit être déterminé lorsque l'objet est instancié, c'est-à-dire à l'exécution.

Smalltalk étant non typé, si j'écris :

```
monObjet afficheToi
```

rien ne permet de savoir quel est le type de **monObjet** et s'il sait répondre à la méthode **afficheToi**. La vérification se fera à l'exécution. Ce code est complètement polymorphique.

Le « look-up » qui consiste à rechercher la bonne méthode à appeler n'est pas performant. Pour pallier à ce problème, on met en place le typage. On peut utiliser le mot clé 'final'. Mais on fige les comportements à la compilation et on déroge à un principe objet.

Ce qui est bien avec les interfaces, c'est qu'on peut rajouter de nouveaux concepts comme « le couplage faible ».

Vous avez plusieurs couches dans votre application, notamment une couche présentation et une couche métier, parce que vous voulez bien faire.

Dans votre couche de présentation (développée avec struts), vous voulez appeler une classe de façade de la couche métier.

Exemple :

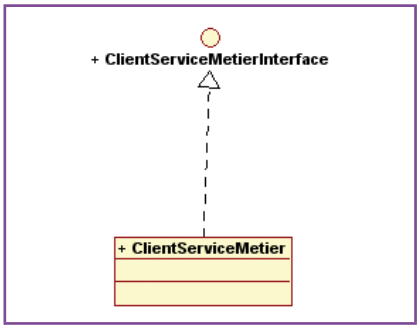
Dans mon action struts j'ai la méthode suivante :

```
public void rechercherClient(ActionForm form) {
   boolean retour = controlesDeSurface();
   if (retour) {
     String nom = form.get("nom");
     String prenom = form.get("prenom");
     List<Personne> clients = ClientServiceMetier.
rechercherClient(nom, prenom);
     //...
   }
}
```

Ma classe **ClientServiceMetier** ne contient que des méthodes statiques. Nous avons fait comme cela pour des raisons de performances. Mais c'est mal, car les deux couches de mon application sont fortement liées.

Les bonnes pratiques recommandent d'utiliser une interface. Comme cela demain, vous pourrez remplacer l'implémentation métier par une autre. C'est ce qu'on appelle le couplage faible.

On va donc remplacer la classe statique par une instance de **ClientServiceMetier** qui implémente une interface **ClientServiceMetierInterface**.



Pour instancier ma classe **ClientServiceMetier**, je passe par une factory (la boucle est bouclée).

```
public void rechercherClient(ActionForm form) {
   boolean retour = controlesDeSurface();
   if (retour) {
    String nom = form.get("nom");
    String prenom = form.get("prenom");
    ClientServiceMetierInterface clm =
(ClientServiceMetierInterface)
      ServiceMetierFactory.getService("ClientService
Metier");
    List<Personne> clients = clm.rechercherClient(nom,
prenom);
      //...
  }
}
```

Ainsi, je pourrais avoir des implémentations métier différentes. La factory saura m'instancier le bon objet qui fera la recherche de clients.

Donc pour me permettre demain de remplacer la couche métier (ce que je ne ferai probablement jamais) je procède à une inflation de mon code en ajoutant la

factory, une interface et de l'instanciation d'objet là où je n'en avais pas besoin.

Pour pallier au problème de performance posé par cette instanciation, je peux maintenant, grâce à ma factory, utiliser un cache ou un pool... encore de la complexité supplémentaire.

Voici tout un concept construit pour pallier à l'absence de métaclasse et au fait que java est fortement typé.

En Smalltalk, le lien entre les couches est, de base, faible...

4 Troisième exemple : les aspects

Je souhaite avoir un moyen non intrusif pour auditer mon code. C'est-à-dire, sans rien changer à mon programme, je souhaite savoir combien de temps on passe dans chaque méthode. Nous avons une solution : les aspects !

C'est formidable. Je fais un bout de code qui trace la durée d'exécution. J'écris où je veux que mon code s'insère (avant / après) et le compilateur se charge de tout.

Mais il y a un double inconvénient :

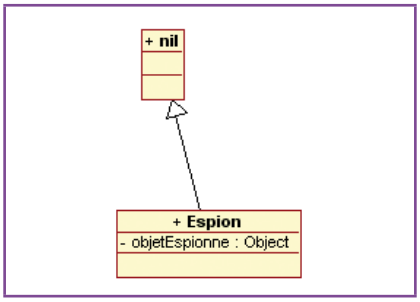
- le code que je vois ne correspond pas au code compilé (le .class et . java ne sont pas en phase), il vaut mieux le savoir ;
- il y a un coût en performance.

Pourquoi est-ce un artifice ?

Et bien en smalltalk (pour ne pas le citer), je peux faire autrement, avec le langage de base.

Je fais une classe 'Espion' qui n'hérite de rien (**nil**) au lieu d'hériter d'Object. Cet objet contient en variable d'instance l'objet espionné.

Comme il n'hérite de rien, il ne sait répondre à aucun message. En smalltalk, tous les messages incompris renvoient vers la méthode **doesNotUnderstand :**. Je redéfinis cette dernière pour faire effectuer ma trace et invoquer la vraie méthode de l'objet espionné.




```
Espion>>doesNotUnderstand: aMessage
  |retour|
  "code avant"
  Retour := self objetEspionne perform: aMessage
selector withArguments: aMessage arguments.
  "code apres"
  ^retour
```

En Smalltalk, l'invocation (l'appel au **perform**) n'est pas plus coûteuse que l'appel direct, car c'est le même mécanisme qui est en œuvre (ce qui n'est pas le cas en java).

Enfin, dans ma classe Espion je fais une méthode de classe **espionne : unObjet**

```
Espion class>>espionne: anObject
| e |
e := self new: anObject.
e become: anObject.
^ anObject
```

La méthode become, permet d'intervertir les deux pointeurs... Magique, non ?

Ce qui donne à l'exécution :

```
| chat |
chat := Chat new.
Espion espionne : chat.
chat print.
```

L'appel à la méthode **print** (ou n'importe quelle autre méthode du chat) entraîne l'exécution de mon code avant et après.

5 Les singletons :

Le design pattern singleton est assez largement utilisé. Pourquoi ? À quoi sert une instance unique ? En général, c'est pour faire du « polymorphisme de classe ».

Les méthodes et les variables statiques ne sont pas des méthodes et des variables de classes. Il est impossible de faire du polymorphisme en static. Rappelez-vous, il n'y a pas de métaclasse en java. Si j'ai besoin d'un compteur d'objets par classe je ne peux pas le gérer une fois pour toutes en static dans ma super classe. Toutes les sous-classes pointeront vers la même variable. Pour répondre à ce problème, la pratique courante est de faire un singleton. Une instance de singleton par compteur. En fait, en y regardant de plus près, si on avait des métaclasses, on ne ferait plus de singleton. Le besoin d'instance unique est en fait très rare. Donc les singletons sont un artifice de plus.

6 Qu'en conclure ?

Java paraît relativement simple à appréhender au départ . Mais pour finir, on se rend compte qu'il est un langage très compliqué, pleins d'exceptions et d'artifices permettant de pallier au fait qu'il n'est pas tout objet. Est-ce un mal ? Tout est une question de point de vue.

Le but de l'objet était d'avoir un langage simple permettant de programmer tel qu'on parle, en langage naturel (fenêtre **afficheToi**, objet **initialiseToi**, objet **faitCaAvec :unAutreObjet**, ...). Mais en tout cas, ce but n'est pas atteint par java.

En java, le langage est complexe et absolument pas naturel. Il est performant, mais à quel prix ?

Les concepts du paradigme objets (l'instanciation, le typage dynamique, l'invocation...) ne sont intrinsèquement pas performants. En revanche, ce sont des concepts puissants.

Vouloir les optimiser n'est-ce pas renier ce qui fait leur force ? ■

CONFIGURATION D'UN POSTE DISKLESS VIA IPXE ET NFSROOT

par Denis Bodor [diskless baby, yeah !]

L'informatique moderne est amusante. Alors que nos données s'échappent et se dispersent dans le « klaoude », un royaume étrange où tout est à la fois partout et nulle part, disponible et sûr (soi-disant), la taille des disques et autres supports de stockage ne cesse de croître, aussi bien sur serveur (bien sûr), les PC Desktop, que les laptops ou les smartphones. Le moment où on nous vantera les mérites de l'OS dans les nuages approche à grands pas. Pourquoi ? Simplement parce que c'est techniquement disponible depuis des années et il ne s'agira là aussi que d'un ensemble réchauffé et savamment marketé. La preuve, vous pouvez déjà le faire...

Trêve de plaisanterie. Cet article ne porte pas sur une quelconque idée pseudo-révolutionnaire, mais sur un retour d'expérience fort technique et fort pragmatique. Tout commence avec un besoin et un cahier des charges très simple : mettre en place une plateforme d'expérimentation pour le développement de pilotes de périphériques PCI (et plus tard PCI Express). La motivation primaire est de développer à la fois l'aspect matériel PCI via un devkit CPLD Altera Max II (DK-MAXII-1270N) et la partie noyau Linux, de manière à appréhender cette technologie de manière holistique et la plus complète possible. Pour ce faire, il me fallait disposer d'une nouvelle machine/configuration, rebootable à souhait, économique et la plus souple possible. Ceci prit la forme d'une simple carte mère mini-ITX Intel D425KT *refurbished* équipée d'un processeur Atom D425 à laquelle ont été ajoutés 2 Go de RAM et un bloc d'alimentation ATX de taille réduite. Pas de fioritures, pas de ventilation active, pas de boîtier et surtout pas de disque, fût-il USB.

La bonne surprise à réception de la bête s'est faite jour avec un petit tour très classique dans le BIOS : on me propose le choix entre un boot réseau PXE et gPXE. GPXE ? Kesako ?

Bien que la plupart des BIOS soient en mesure de démarrer sur le réseau via PXE (prononcer « pixie ») pour *Preboot eXecution Environment*, les fonctionnalités offertes sont assez limitées. PXE est un environnement décrit dans une partie des spécifications du framework WfM (*Wired for Management*) proposées par Intel. On y trouve également la définition du standard WOL (*Wake-On-LAN*) permettant le démarrage (ou le réveil) d'un PC à distance via le réseau. Depuis, WfM est devenu obsolète et remplacé par IPMI pour *Intelligent Platform Management Interface* dont le nom laisse penser que quelqu'un avait omis d'intégrer l'élément « intelligent » dans WfM... Et en effet, il est généralement entendu que PXE est tout bonnement impossible à implémenter en suivant strictement les spécifications d'origine et que, de ce fait, la plupart des fonctionnalités de boot PXE intégrées en OEM sont plus ou moins boguées. Des exemples de problèmes inhérents aux spécifications sont régulièrement exposés lors des *talks* des développeurs Etherboot et gPXE (YouTube est votre ami).

Depuis très longtemps, le mécanisme de démarrage de machines *diskless* repose sur deux options : d'une part celle proposée par le fabricant de l'interface Ethernet sous la forme d'un code intégré dans une mémoire (bootROM) placée sur la carte (soudée ou en option) et d'autre part Etherboot, un code GPL pouvant être copié dans une bootROM, sur une disquette, dans une partie de la flash BIOS, etc. Les plus anciens se souviendront certainement des UVPROM à placer sur les cartes ISA compatibles NE2000, tout ceci n'est pas nouveau.

Avec l'arrivée de PXE, une scission est apparue au sein du projet Etherboot. D'un côté se trouvaient les développeurs tenant à conserver les choix techniques déjà en place car meilleurs que ceux présents dans les spécifications PXE et,

de l'autre, les développeurs préférant opter pour une extension d'Etherboot vers PXE en apportant, de plus, un certain nombre d'améliorations hors des spécifications. En d'autres termes, deux directions opposées déchiraient le projet. Ainsi est né gPXE au sein de la communauté Etherboot avant que finalement, les choses ne s'enveniment vraiment et que gPXE quitte définitivement le giron d'Etherboot et ce, semble-t-il, dans une ambiance relativement électrique. La FAQ d'iPXE précise en effet la raison du changement de nom troquant le « g » pour un « i » : « *iPXE is developed by the people who originally developed gPXE (which evolved from Etherboot). Unfortunately, the gpxe.org and etherboot.org domains are owned by an individual who wishes to exercise a high degree of control over the project and the codebase[...]* ». Un peu rude, mais généralement, un fork issu de développeurs œuvrant au sein du projet original est rarement quelque chose qui se passe dans le plus grand calme...

Revenons à mes moutons. Embarqué dans ma carte mère se trouvait donc, pré-installé, un code open source, une implémentation améliorée de PXE qui ne demandait qu'à être explorée. Une chose en amenant une autre, l'intérêt pour gPXE/iPXE en arriva au point de vouloir l'utiliser sur d'autres machines ne disposant pas de la fonctionnalité en standard et c'est précisément ce que nous allons voir dans le présent article.

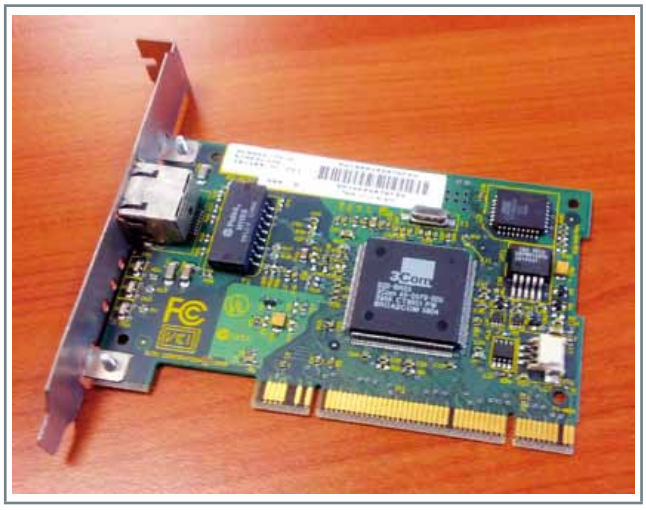
Bien qu'il existe plusieurs solutions pour une configuration diskless, nous nous en tiendrons ici au classique : un boot iPXE et système de fichiers racine en NFS. Parmi les alternatives, nous avons l'ensemble en RAMFS ou encore, la solution du périphérique bloc via le réseau, comme par exemple iSCSI ou NBD (ou encore l'*ATA over Ethernet*).

1 Configuration du boot iPXE

La première étape de la configuration consiste à doter l'interface réseau de la capacité à récupérer des binaires via le réseau afin de les exécuter localement. De manière générale, il s'agit souvent d'un noyau Linux et de l'image initrd qui l'accompagne, mais techniquement rien ne vous interdit de vouloir booter d'autres OS comme un *BSD, un FreeDOS, ou même un système plus récent de la firme de Redmond.

La carte mère Intel dispose déjà d'un support gPXE et ne nécessite donc pas de manipulations complémentaires. Mais la plupart des machines récentes ou moins récentes utilisent leur propre implémentation PXE directement placée dans le BIOS (voire aucune implémentation, car simplement sans interface Ethernet intégrée). Il est surprenant de constater ô combien il peut être difficile de trouver une carte réseau PCI neuve auprès des détaillants de matériel. En dehors des cartes dites « serveur » dont le prix dépasse souvent les 150€, la prolifération des interfaces 10/100 bon marché est de l'histoire ancienne. Trouver une carte PCI disposant d'un emplacement pour bootROM est une tâche difficile. Trouver une bootROM est presque impossible et se résume souvent à abandonner le monde de l'informatique pour celui de l'électronique afin d'acquérir une EEPROM parallèle vierge compatible. Mieux vaut finalement opter pour un achat d'occasion, bien plus économique. La course aux nouvelles technologies, qui devient une stupide fuite en avant, n'a pas que du bon. Les interfaces Ethernet PCI ne sont pas les seules concernées, essayez donc de trouver un disque ou un lecteur CDROM EIDE, une imprimante matricielle ou un modem. Et je ne parle même pas des consommables... Demandez un ruban d'imprimante ou un câble DB25 à un vendeur dans la vingtaine, vous verrez, c'est assez amusant.

Dans le cadre des expérimentations autour de cet article, deux modèles de carte Ethernet ont été utilisés : une 3c905c de 3com disposant d'une bootROM soudée intégrée et une carte bien plus courante/économique (en son temps), une Realtek RT8139 possédant un emplacement pour bootROM sur lequel pourrait être enfichée une EEPROM Flash de 1Mb (128Ko) Atmel AT29C010A (~15€ les 10 sur eBay).



La carte Ethernet 3com 3c905C-TX/TX-M est une interface de choix pour créer un poste diskless disposant d'un emplacement PCI. Il existe bien entendu plusieurs modèles de 3c905, mais la plupart se voient équipés directement d'une EEPROM de boot soudée.

Produire le code à placer dans la bootROM n'est pas vraiment nécessaire. Il est parfaitement possible de générer en ligne, sur rom-o-matic.net, une image (disquette, CDROM, USB) ou une ROM et même de personnaliser les options souhaitées. Cependant, vous conviendrez avec moi que même si cela est fort pratique, c'est beaucoup moins amusant et cela n'offre pas la possibilité d'étudier le code et/ou de le modifier. Nous allons donc utiliser la méthode consistant à compiler et construire notre ROM iPXE depuis les sources les plus récentes.

Première étape, récupérer les sources via le serveur Git :

```
% git clone git://git.ipxe.org/ipxe.git
Cloning into ipxe...
remote: Counting objects: 38691, done.
remote: Compressing objects: 100% (10006/10006), done.
remote: Total 38691 (delta 28803), reused 36788 (delta 27124)
Receiving objects: 100% (38691/38691), 9.44 MiB | 1.53 MiB/s, done.
Resolving deltas: 100% (28803/28803), done.
```

Après avoir vérifié les dépendances de compilation, on s'empressera de construire l'ensemble avec les options par défaut via un simple **make** :

```
% cd ipxe/src/
% make -j 15
[...]
To create a bootable floppy, type
    cat bin/ipxe.dsk > /dev/fd0
where /dev/fd0 is your floppy drive.  This will erase any
data already on the disk.

To create a bootable USB key, type
    cat bin/ipxe.usb > /dev/sdX
where /dev/sdX is your USB key, and is *not* a real hard
disk on your system.  This will erase any data already on
the USB key.

To create a bootable CD-ROM, burn the ISO image
bin/ipxe.iso to a blank CD-ROM.

These images contain drivers for all supported cards.  You
can build more customised images, and ROM images, using
    make bin/<rom-name>.<output-format>
```

Comme l'indiquent les messages post-compilatoires, nous obtenons plusieurs éléments directement utilisables :

- **bin/ipxe.dsk** : une image de disquette,
- **bin/ipxe.usb** : une image à copier sur clé USB,
- **bin/ipxe.iso** : et une image ISO prête à être gravée sur un CDR.

Ces images volumineuses disposent de la prise en charge de toutes les interfaces supportées par le projet. Quelle que soit votre carte, si elle est compatible, elle pourra être utilisée. Tout ceci est très intéressant pour procéder aux tests, mais d'un point de vue purement pratique, le fait d'utiliser un lecteur (CD ou disquettes) ou un support USB n'est guère attrayant.

Pour produire une image de ROM, il faut avant toutes choses savoir précisément la marque, le type et le modèle d'interface dont on dispose. Pour ce faire, rien de plus simple que d'utiliser **lspci** (via un boot sur un LiveCD ou insertion de la carte dans un autre PC). Voici ce que nous obtenons pour la carte 3com :

```
% lspci
[...]
00:18.4 Host bridge: Advanced Micro Devices [AMD]
 Family 10h Processor Link Control
01:0a.0 Ethernet controller: 3Com Corporation
  3c905C-TX/TX-M [Tornado] (rev 74)

% lspci -n -s 01:0a.0
01:0a.0 0200: 10b7:9200 (rev 74)
```

La seconde occurrence de la commande nous permet, via les options **-n -s**, d'obtenir le Vendor ID et le Product ID du périphérique via une désignation sous la forme **<numéro du bus>:<numéro du périphérique>.<numéro de fonction>**. On utilisera alors ces deux valeurs pour composer le nom de la cible de compilation avec **make** :

```
% make bin/10b79200.rom
  [LD] bin/10b79200.rom.tmp
  [BIN] bin/10b79200.rom.bin
  [ZINFO] bin/10b79200.rom.zinfo
  [ZBIN] bin/10b79200.rom.zbin
  [FINISH] bin/10b79200.rom
rm bin/10b79200.rom.zinfo bin/10b79200.rom.zbin bin/10b79200.rom.bin
```

Le fichier obtenu, **bin/10b79200.rom**, est l'image directement utilisable pour flasher la bootROM de la carte. Dans le cas de la 3c905c, il n'est pas nécessaire de passer par les outils du fabricant généralement fournis pour DOS (oui oui, MS/DOS) sous la forme d'un **.EXE**.

Pour écrire nos données en EEPROM, nous pouvons faire usage du fantastique **flashrom**, un outil capable de lire, d'écrire et de vérifier le contenu de pas moins de 300 composants, aussi bien via des périphériques dédiés (USB, parallèle, série, etc.) que via les fonctionnalités offertes par les cartes mères (quelques 400 modèles) et les périphériques PCI (environ 50 modèles). Parmi cette pléthore de fonctionnalités on trouve le support de l'EEPROM Flash équipant la 3c905c, ainsi que celui de la carte elle-même. Inutile donc de devoir mettre en œuvre de quelconques manipulations hasardeuses et un périphérique dédié, une simple ligne de commandes sous GNU/Linux suffit. On commence ainsi par déterminer le type d'EEPROM et par la même occasion, vérifier la bonne prise en charge du matériel via :

```
% flashrom -p nic3com
Calibrating delay loop... OK.
Found Atmel flash chip "AT49BV512"
  (64 kB, Parallel) on nic3com.
No operations were specified.
```

Une puce Atmel équipe la carte 3c905 et un coup d'œil à cette dernière nous confirme la réussite de la détection. On enchaîne donc ensuite sur la réécriture des données qui y sont contenues :

```
% flashrom -p nic3com -w 10b79200.rom
Calibrating delay loop... OK.
Found Atmel flash chip "AT49BV512"
  (64 kB, Parallel) on nic3com.
Reading old flash chip contents... done.
Erasing and writing flash chip... Erase/write done.
Verifying flash... VERIFIED.
```

RT8139 FreeDOS flash

La carte à base du chip Realtek RTL8139C nous a posé énormément de problèmes avant de finalement être abandonnée au profit de l'utilisation exclusive d'interfaces un peu plus « haut de gamme » comme 3COM et Intel. La très grande majorité des cartes bas de gamme intégrant ce composant ne disposent pas d'emplacement pour bootROM. Celles qui le proposent sont capables d'accueillir une mémoire au format DIL28. Or, la mémoire la plus couramment utilisée en guise de bootROM est l'Atmel AT29C010A.

Nous avons joué de malchance dans ces expérimentations. Le vendeur chinois (eBay) a envoyé le mauvais format de composant (PLCC au lieu de DIL) et c'est en essayant de l'adapter via un hideux assemblage qu'il s'est avéré que l'AT29C010A utilise 32 broches et ce, au format PLCC et DIL. Pourtant, le programme DOS **RTFLASH.EXE** est bien à même de reconnaître ce type de composant. Il s'avère finalement que la majorité des cartes à base de RTL8139 disposent d'un emplacement ne permettant pas d'écrire dans la mémoire, mais uniquement de l'utiliser (la lire). Il faut ainsi mettre en œuvre non pas une EEPROM flash, mais une UVPROM de type 27C512, bien plus difficile à programmer car nécessitant un équipement particulier.

La raison de cette profusion de cartes « handicapées » provient du coût supplémentaire induit par la présence de la fonctionnalité d'écriture. Le Realtek 8319 était le composant le plus utilisé pour produire des cartes à prix réduit et les fabricants de clones, sur ce marché très agressif, essayaient par tous les moyens de réduire les coûts de production.

Si vous ne voulez pas être déçu, nous vous recommandons de bien vous assurer de la présence d'un emplacement DIL32 avant achat ou, plus simplement, de choisir une 3C905C d'occasion qui ne vous coûtera pas vraiment plus cher.

C'est tout ! Notre carte Ethernet est maintenant équipée d'un code de boot iPXE et de toutes ses fantastiques fonctionnalités. J'ai remarqué qu'en fonction du compilateur et de la distribution utilisée, la compilation d'iPXE peut déboucher sur la création d'un fichier **.rom** d'une taille inadéquate. Dans ce cas, **flashrom** ne manquera pas de nous signaler le problème et ainsi d'abandonner toute tentative d'écriture :

```
% flashrom -p nic3com -w 10b79200.rom
[...]
Error: Image size doesn't match

% ls -ln bin/10b79200.rom
-rw-rw-r-- 1 0 0 66048 juin 18 11:58 bin/10b79200.rom
```

Cette image de 66048 octets en lieu et place des 65536 disponibles dans l'AT49BV512 a été obtenue avec la même version Git du projet, les mêmes options, mais sur une récente distribution Ubuntu 13.04, alors qu'aucun problème de ce type n'est apparu avec ma très chère Debian que j'utilise quotidiennement (on va encore dire que je suis mauvaise langue, mais je ne fais que relever ce détail). Il faut savoir que l'une des contraintes propres à Etherboot et gPXE/iPXE est celle de l'optimisation du volume du binaire compilé. Ceci implique des difficultés dans la chasse aux bugs, un impact important des options de compilation par défaut et rend également difficile l'ajout de fonctionnalités. Chaque octet compte et la moindre option est importante.

L'EEPROM flash parallèle équipant notre 3C905 est une Atmel AT49BV512 de 64Ko parfaitement prise en charge par flashrom. La programmation du composant est un jeu d'enfant, même si l'espace disponible ne permet pas d'embarquer absolument toutes les fonctionnalités de gPXE/iPXE.

Vous pouvez réduire la taille du binaire en désactivant certaines fonctionnalités en éditant le fichier **config/general.h** et en passant à **#undef** les éléments que vous n'utiliserez pas. C'est d'ailleurs une bonne manière de se rendre compte de l'étendue de ce que propose iPXE :

- Obtention de fichiers via TFTP, HTTP, FTP, AoE, iSCSI, Fibre Channel...
- Prise en charge de quelques interfaces Wifi avec support WEP, WPA et WPA2,
- Fonctionnalités de scripting avancé (tests/conditions, variables, menu, etc.),
- Support VLAN,
- Résolution DNS,

- Interface de configuration interactive,
- Support de nombreux formats d'images (ELF, Multiboot, PXE, bzImage, EFI, etc.),
- Support IPv6,
- Etc.

Le lecteur pertinent aura sans doute remarqué que tout cela permettra, par exemple, des choses très exotiques comme le fait de booter *over Wifi* avec des éléments se trouvant sur un serveur web dans la Toile. Quand je vous parlais de klaoude...

Désactivez quelques fonctionnalité et relancez **make**. 65536 est votre objectif ; si le code est plus petit, le fichier résultant sera *paddé* avec **0xff**. Un petit coup de **hexdump** vous permettra ainsi de savoir où vous en êtes :

```
% hexdump -v bin/10b79200.rom
[...]
000fe70 0773 03de 38aa c678 12b3 bd1b 7348 6f3a
000fe80 26cc a753 770d 7dfb 32db 19cb d45d c4d9
000fe90 1b26 5774 5f2f d8e0 dd75 230b 6bab 0734
000fea0 8107 76cf 54b0 27d1 f212 4b2b b470 d34d
000feb0 0e74 07d8 3682 2c91 14d7 34d5 63a0 23fc
000fec0 0023 8e00 073a 0bcb 0004 0000 0000 8000
000fed0 ffff ffff ffff ffff ffff ffff ffff ffff
000fee0 ffff ffff ffff ffff ffff ffff ffff ffff
000fef0 ffff ffff ffff ffff ffff ffff ffff ffff
000ff00 ffff ffff ffff ffff ffff ffff ffff ffff
000ff10 ffff ffff ffff ffff ffff ffff ffff ffff
000ff20 ffff ffff ffff ffff ffff ffff ffff ffff
000ff30 ffff ffff ffff ffff ffff ffff ffff ffff
000ff40 ffff ffff ffff ffff ffff ffff ffff ffff
000ff50 ffff ffff ffff ffff ffff ffff ffff ffff
000ff60 ffff ffff ffff ffff ffff ffff ffff ffff
000ff70 ffff ffff ffff ffff ffff ffff ffff ffff
000ff80 ffff ffff ffff ffff ffff ffff ffff ffff
000ff90 ffff ffff ffff ffff ffff ffff ffff ffff
000ffa0 ffff ffff ffff ffff ffff ffff ffff ffff
000ffb0 ffff ffff ffff ffff ffff ffff ffff ffff
000ffc0 ffff ffff ffff ffff ffff ffff ffff ffff
000ffd0 ffff ffff ffff ffff ffff ffff ffff ffff
000ffe0 ffff ffff ffff ffff ffff ffff ffff ffff
000fff0 ffff ffff ffff ffff ffff ffff ffff ffff
0010000
```

Le fichier **.tmp.map** accompagnant l'image ROM vous apportera également quelques informations intéressantes en termes de fonctionnalités embarquées.

Une fois l'opération de flashage réussie, il ne vous reste plus qu'à placer la carte dans la machine de destination, sans oublier de faire un tour dans la configuration du BIOS afin de reconfigurer l'ordre des périphériques de boot. Cela peut paraître logique, mais le fait que la bootROM iPXE donne systématiquement l'opportunité d'interrompre le processus de démarrage via CTRL+B peut être source de confusion. Ce n'est pas parce que vous voyez le message iPXE et qu'une utilisation interactive fonctionne, que le périphérique sera forcément celui utilisé en premier pour booter (ceci m'a coûté une bonne 1/2 heure avant d'avoir subitement un éclair de lucidité).

2 Côté serveur : DHCP et TFTP

BOOTP, initialement utilisé pour ce type de démarrage, n'est plus du tout d'actualité. De nos jours, grâce aux ressources grandissantes et aux différentes améliorations, les clients diskless peuvent se passer de l'ersatz BOOTP et utiliser DHCP. Il est fort probable que vous ayez déjà en place un serveur DHCP sur votre réseau tant personnel que professionnel. Cette probabilité s'étend même au fait qu'il s'agit très certainement d'ISC DHCP. Si tel n'est pas le cas, son installation est l'affaire de quelques secondes et sa configuration d'une poignée de minutes.

```
La configuration du serveur ISC DHCP, concernant notre hôte équipé de
la 3c905c, se résume à ceci host test3c905C {
  hardware ethernet 00:01:02:07:6f:5a;
  fixed-address 192.168.10.145;
  option routers 192.168.10.1;
  option domain-name-servers 8.8.8.8 ;
  filename "ipxe/base.ipxe";
  next-server 192.168.10.166;
}
```

Tout à fait classiquement, on retrouve l'attribution de l'adresse IP, la configuration du réseau, de la passerelle par défaut, du DNS, etc. La partie importante pour le boot tient dans les deux dernières lignes en précisant un fichier à proposer au client via **filename** et un serveur à contacter pour ce faire, via **next-server**. Nous sommes ici dans une démarche tout à fait classique, où le serveur DHCP fournit la configuration IP et un serveur TFTP (*Trivial File Transfer Protocol*) est chargé de délivrer le fichier. En temps normal, le fichier utilisé est **pxelinux.0** ou éventuellement **gpxelinux.0**, livré avec Syslinux, un ensemble de briques et de solutions de boot locales, distantes, sur support CD, etc.

Ici, avec iPXE, nous n'avons pas besoin d'un second bootloader pour charger d'autres éléments comme c'est le cas avec les simples implémentations PXE. Nous spécifions au contraire un fichier **base.ipxe** placé dans le répertoire **ipxe** de notre serveur TFTP. Ce fichier n'est pas un binaire mais un script, un simple fichier texte que iPXE va récupérer, lire, interpréter et exécuter.

L'installation du serveur TFTP est, tout comme pour le DHCP, quelque chose qui n'est pas nouveau. En effet, TFTP offre un grand intérêt dans l'essai, le développement et l'expérimentation de systèmes embarqués, puisqu'il s'agit d'un des protocoles quasi-systématiquement activés dans U-Boot si la plateforme dispose d'un support réseau. Personnellement, j'utilise le serveur **atftpd** (via le package du même nom sous Debian GNU/Linux) et non **tftpd** ou **tftpd-hpa**. Ce serveur peut fonctionner seul (*standalone*) ou via le super-serveur Xinetd, méthode que je préfère généralement.

Ainsi, un simple fichier de configuration **/etc/xinetd.d/tftp** permet la mise en place du service :

```
service tftp
{
  disable     = no
  socket_type = dgram
  protocol    = udp
  port        = 69
  wait        = yes
  user        = root
  server      = /usr/sbin/atftpd
  server_args = /home/denis/EMB/TFTP
}
```

Dès lors, le serveur TFTP est accessible via le port udp/69 et sa racine se trouve dans mon **~/EMB/TFTP**. Pour procéder à un premier test, il suffit alors de créer le répertoire **ipxe** et de le peupler :

- **vmlinuz-3.9-1-686-pae** : une image d'un noyau provenant directement du **/boot** d'une distribution Debian,
- **initrd.img-3.9-1-686-pae** : l'image initrd correspondante,
- **base.ipxe** : notre script iPXE contenant alors :

```
#!ipxe
echo "Greetings! Hit Ctrl-C to bail out"
echo "Loading kernel image"
kernel tftp://192.168.10.166/ipxe/vmlinuz-3.9-1-686-pae
  boot=nfs nfsroot=192.168.10.166:/home/denis/EMB/NFSROOT1
  ip=dhcp console=tty nomodeset rw
echo "Loading initrd image"
initrd tftp://192.168.10.166/ipxe/initrd.img-3.9-1-686-pae
boot
```

On « sha-bang » simplement le fichier texte avec **ipxe** afin que l'ensemble soit considéré comme un script et iPXE s'occupe du reste. Notez que l'entrée **kernel** est ici présentée sur plusieurs lignes, mais que chaque commande/directive, dans les faits, est sur une seule et unique ligne, faute de quoi vous obtiendrez une erreur. **kernel** et **initrd** se passent ici d'explication, c'est quelque chose de commun à l'ensemble des booloaders. **echo** nous permet d'ajouter des messages et **boot** provoque le démarrage. **boot**, **chain** ou **imgexec** sont synonymes. iPXE et ses commandes sont relativement souples. Ici, avec **boot**, nous aurions également pu nous passer de la ligne **kernel** en chargeant uniquement l'image **initrd** et en faisant **boot tftp://192.168.10.166/ipxe/vmlinuz-3.9-1-686-pae boot=nfs[...]**.

Notez que, de la même manière, **kernel** est synonyme de **imgselect** et **imgload**, tout comme **initrd** est identique à **imgfetch** et **module**. Le premier argument passé aux deux directives est une URI. Nous faisons usage ici de **tftp:**, mais ceci fonctionnerait tout aussi bien avec **http:** ou **ftp:**. Mieux encore, les URI en question, si elles sont utilisées dans un script, peuvent se limiter au nom de fichier. iPXE va alors chercher l'objet relativement à l'URI du script lui-même et faire de notre **base.ipxe** quelque chose de plus léger.

3 Rootfs en NFS

À ce stade, notre machine diskless est en mesure de démarrer le noyau et d'initialiser un système basique en initramfs, mais ce démarrage s'interrompt rapidement, puisqu'il n'existe pas encore de système de fichiers racine à monter et utiliser. Nous allons maintenant nous attacher à la configuration de cet élément. La première chose à faire est d'initialiser un système, une distribution, dans un répertoire de la machine serveur, que nous exporterons en NFS par la suite. Pour ce faire, Debian propose une solution prenant la forme de l'utilitaire **debootstrap**. Tout ce que nous avons à faire est de créer le répertoire cible (ici **~/EMB/NFSROOT1**) et d'utiliser :

```
% sudo debootstrap testing /home/denis/EMB/NFSROOT1
I: Retrieving Release
I: Retrieving Packages
I: Validating Packages
I: Resolving dependencies of required packages...
I: Resolving dependencies of base packages...
[...]
I: Validating aptitude
I: Retrieving aptitude-common
I: Validating aptitude-common
I: Retrieving libattr1
I: Validating libattr1
I: Retrieving base-files
I: Validating base-files
I: Retrieving base-passwd
I: Validating base-passwd
I: Retrieving bash
[...]
I: Configuring aptitude...
I: Configuring iputils-ping...
I: Configuring tasksel...
I: Configuring tasksel-data...
I: Configuring libc-bin...
I: Base system installed successfully.
```

On précise simplement la version de la distribution, ici **testing** (et non **stable**), ainsi que le répertoire de destination. **debootstrap** se charge alors de récupérer les différents paquets et leurs dépendances et de les installer dans **~/EMB/NFSROOT1**, comme s'il s'agissait de la racine du système. À présent, ce répertoire contient un système presque complet. Pour finaliser sa configuration, nous pouvons « chrooter » dans le répertoire après avoir copié le **resolv.conf** du système hôte, afin de pouvoir résoudre les noms des machines sur le grand Internet :

```
% cp /etc/resolv.conf ~/EMB/NFSROOT1/etc/
% chroot ~/EMB/NFSROOT1
```

Dès lors, tout se passera comme si nous étions en train d'utiliser le nouveau système. Nous pouvons ainsi mettre à jour la liste des paquets et créer un nouvel utilisateur :

```
# apt-get update
[...]
Fetched 4243 kB in 14s (295 kB/s)
Reading package lists... Done

# adduser  denis
Adding user `denis' ...
Adding new group `denis' (1000) ...
Adding new user `denis' (1000) with group `denis' ...
Creating home directory `/home/denis' ...
Copying files from `/etc/skel' ...
Enter new UNIX password: **************
Retype new UNIX password: **************
passwd: password updated successfully
Changing the user information for denis
Enter the new value, or press ENTER for the default
        Full Name []: Denis
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n]
```

Nous installons également quelques paquets parmi lesquels, par exemple, **locales**, nous permettant d'éviter d'être dérangés par des avertissements concernant les pages de code absentes ou non configurées. Un judicieux **dpkg-reconfigure locales** nous permettra de corriger tout cela. Rien n'est vraiment critique à ce stade, étant donné que le système de base est déjà fonctionnel. Il s'agit plutôt d'installer une majorité de choses immédiatement (**vim**, **build-essential**, **mc**, **ctags**, etc.), en local (ce n'est qu'un environnement chrooté), pour ne pas avoir à le faire ensuite, sur la machine diskless, avec les ralentissements inhérents au système de fichiers NFS. On pourra également copier les éléments « personnels » de l'utilisation dans son **$HOME** (configuration de Vim, profil Bash, etc.). N'oubliez pas cependant que vous êtes **root** et que ces fichiers copiés devront appartenir à l'utilisateur cible. Vérifiez UID/GID numériquement, car la correspondance avec un nom d'utilisateur et de groupe peut être fausse vue de l'extérieur du chroot. N'oubliez pas également d'installer et configurer **sudo**, ainsi que **console-data**/**console-setup** pour respectivement faciliter vos manipulations sur le système une fois booté et disposer d'une configuration console avec un mappage clavier adéquat (il est probable que la configuration de ce dernier élément échoue en chroot du fait de l'absence de montage de **/proc** et de devfs ; vous pourrez finaliser l'installation une fois la machine diskless démarrée). Enfin, éditez le fichier **/etc/hostname** qui, par défaut, contiendra le nom de la machine hôte, pour définir un nom différent.

Une fois tout cela en place, on quittera l'environnement chrooté avec un CTRL+D et on se penchera sur la configuration du serveur NFS. Très simplement, cela se résume à l'installation du service via le paquet **nfs-kernel-server** sur Debian, puis à l'édition du fichier **/etc/exports**. Dans notre cas, son contenu se résumera à ceci (sur une ligne toujours) :

```
/home/denis/EMB/NFSROOT1
   192.168.10.145(rw,async,no_root_squash,subtree_check)
```

On mettra ensuite à jour la liste des exportations avec :

```
% sudo exportfs -rv
exporting 192.168.10.145:/home/denis/EMB/NFSROOT1
```

Bien entendu, il est également possible de redémarrer purement et simplement le serveur NFS, mais l'utilisation de **exportfs** est une bonne habitude, car sur une machine hébergeant plusieurs arborescences exportées, il n'est pas question d'arrêter et redémarrer le service globalement. Notez que par défaut c'est NFSv4 qui sera utilisé et, de ce fait, un fonctionnement *over TCP* est bien plus efficace que l'utilisation du protocole non connecté UDP. Si vous rencontrez des problèmes sur un réseau fortement congestionné, modifier l'argument **nfsroot** passé au noyau (via **base.ipxe**) peut éventuellement aider : **nfsroot=192.168.10.166:/home/denis/EMB/NFSROOT1,rsize=8192,wsize=8192,tcp**. L'optimisation des performances de NFS sort du cadre de ce simple article, mais sachez qu'il existe un certain nombre de techniques permettant de réellement optimiser ce type de configuration, au point d'obtenir des performances presque similaires à un rootfs local (j'ai bien dit « presque similaires »).

À présent, tout est fin prêt pour démarrer notre machine diskless qui devrait alors vous fournir un login sans le moindre problème. Il ne vous restera plus, ensuite, qu'à finaliser la configuration et éventuellement installer les éléments complémentaires pour en faire une machine de travail. Notez qu'il peut être intéressant d'installer sur la machine en question un nouveau noyau. En effet, dans l'état, **/boot** est vide et **/lib/modules** également. Le fait de procéder à l'installation complétera non seulement la configuration, mais produira une nouvelle image initramfs/initrd adaptée au matériel, en plus de rendre disponibles les modules qui peuvent être nécessaires pour la prise en charge du matériel « secondaire » ou hotplug. Les paquets **firmware-linux-free** et **firmware-linux-nonfree** sont à installer pour les mêmes raisons. N'oubliez pas, ensuite, de copier les images noyau et initrd sur le serveur TFTP pour le prochain boot.

4 Autres fonctionnalités amusantes d'iPXE

Nous venons de voir une utilisation très classique d'iPXE, très proche de ce que l'on ferait avec un simple bootcode PXE. Mais iPXE dispose de fonctionnalités avancées bien au-delà du support des protocoles réseau comme HTTP pour récupérer les images utiles au boot. Bien qu'offrant des possibilités très intéressantes, le boot HTTP, par exemple, ne

nécessite pas de longues explications puisque cela se résume à désigner, via **filename**, une URL comme http://serveur/rep/image.ext en lieu et place de simplement **ipxe/base.ipxe**. Qui dit « serveur HTTP » peut éventuellement dire « CGI » ou « code PHP » (berk), ce qui offre des options supplémentaires côté serveur pour fournir un script iPXE ou une image de manière dynamique. Mais ce qui nous intéresse ici concerne davantage les fonctionnalités propres à iPXE, comme la possibilité d'utiliser un système de menus.

Considérez, par exemple, le début de script **base.ipxe** suivant :

```
#!ipxe

menu Choose your destiny :
item --key x linux1   Boot GNU/Linux NFS-1   (x)
item --key y linux2   Boot GNU/Linux NFS-2   (y)
item --key s shell    iPXE shell             (s)
item --key e exit     BIOS                   (e)
choose --default linux2 --timeout 5000 target && goto ${target}
[...]
```

La directive **menu**, si l'option est compilée dans la bootrom, nous permet d'afficher un menu (en couleurs SVP !) proposant un choix à l'utilisateur de la machine diskless. Nous avons ici quatre entrées désignant respectivement deux boots via TFTP/NFS, un appel au shell iPXE et une sortie de l'exécution du code de boot (qui en principe découle sur l'utilisation du prochain périphérique de démarrage spécifié dans la configuration du BIOS). La syntaxe utilise la directive **item** suivie d'un nom d'étiquette et d'une chaîne de caractères constituant le texte à afficher à l'utilisateur. **--key** est une option nous permettant de proposer une touche de raccourci. Notez que le mappage clavier est QWERTY à ce moment et qu'il n'existe pas de système permettant d'indiquer automatiquement le raccourci configuré (vous devez le spécifier dans le texte de l'entrée du menu).

L'affichage et l'utilisation du menu sont gérés par la directive **choose** qui ici spécifie un délai de 5 secondes (**--timeout**) après lequel l'étiquette **linux2** est utilisée via la variable **target**. Il ne reste plus ensuite qu'à définir les étiquettes et les instructions associées avec quelque chose comme :

```
:linux1
kernel vmlinuz1 boot=nfs root=/dev/nfs[...]
initrd initrd1.img
boot

:linux2
kernel vmlinuz2 boot=nfs root=/dev/nfs[...]
initrd initrd2.img
boot

:exit
exit

:shell
shell
```

Ainsi, le boot réseau affichera le menu et après un délai de 5 secondes, sans intervention de l'utilisateur, **vmlinuz2** et **initrd2.img** seront démarrés. Le choix de l'utilisateur pourra se faire par sélection dans le menu avec les flèches de direction et validation, ou en utilisant le raccourci (sans validation).

Le fonctionnement de la commande **exit** est dépendante du comportement effectif du BIOS de la machine et il n'est pas certain que ceci découle effectivement sur un boot via support amovible, USB ou disque local. Vous pouvez parfaitement vous retrouver avec un arrêt pur et simple de la procédure de boot. Si iPXE est compilé avec le support SAN, il est alors plus sûr d'utiliser **sanboot –no-describe –drive 0x80** pour démarrer sur le premier disque local de la machine (via une entrée du menu). Dans le cas contraire, la solution peut être de passer le relais à un bootloader comme GRUB4DOS, récupéré via le réseau avec une commande comme **chain grub.exe --config-file="root (hd0,0);chainloader +1"**. Remarquez que le contenu du fichier de configuration est directement spécifié sur la ligne de commandes et que, là encore, on peut parfaitement confier au serveur HTTP (si ce protocole est utilisé pour récupérer **grub.exe**) la configuration de manière dynamique de cette chaîne de caractères.

Enfin, il est important de noter que la gestion des menus n'est présente de manière complète que dans les dernières versions d'iPXE. La bootROM gPXE de la carte mère Intel, par exemple, ne les prend pas en charge, pas plus que les pauses via la directive **sleep**.

Conclusion

Nous sommes arrivés à la fin de cet article et vous avez pu constater qu'iPXE est un fabuleux morceau de code développé par des programmeurs de talent possédant une grande expérience du domaine. Pour quelqu'un comme moi, pour qui la dernière configuration d'un boot réseau mettait en œuvre des cartes ISA compatibles NE2000 et des Pentium, il s'agit d'un fantastique bon en avant. En termes de fonctionnalités, il est amusant de constater que tout en arrivant à faire tenir cela dans quelques 64 Ko, on dispose d'éléments qui devraient être depuis longtemps inclus dans le BIOS vieillissant de nos chers PC. Certes, UEFI pointe le bout de son nez, mais tout comme PXE, il ne s'agit que de spécifications. iPXE dispose déjà de codes et d'un plan pour intégrer et étendre UEFI (http://ipxe.org/efi/vision). Qui sait, peut-être, à terme, les efforts conjugués de projets comme iPXE ou coreboot, avec la participation de fabricants comme Intel, nous permettront de disposer, en standard (comprendre « sans bidouille »), d'un véritable remplaçant open source aux BIOS actuels propriétaires, obscurs et incluant souvent des bugs ? ■

ERGONOMIE DES SYSTÈMES MOBILES

par Jérémie CHAINE & Philippe PRADOS
[Ergonome et Consultant @ OCTO Technology]

Dans la première partie, nous avons commencé à explorer les différents composants ergonomiques des systèmes mobiles. Nous continuons et terminons ici notre voyage.

Pour manipuler les applications, l'utilisateur peut s'appuyer sur des composants graphiques (approche explicite) ou sur sa connaissance d'interactions implicites, dans les gestures. Nous estimons que cette dernière approche doit être réservée aux interactions avancées, car la plupart des utilisateurs les ignorent. Reprenons notre analyse.

1 Les actions de l'utilisateur

1.1 Action Recherche

La recherche est un élément important pour les applications et les OS mobiles. Elle peut être limitée à l'application ou s'étendre à l'ensemble du téléphone.

La recherche peut être localisée à une application ou globale au téléphone. Cette recherche globale permet de retrouver tous les éléments associés à un mot (contacts, réseaux sociaux, notes, documents, web, etc.). Les applications peuvent s'intégrer dans ce mécanisme pour exposer leurs données dans un résultat de recherche, et ainsi inciter l'utilisateur à revenir dans l'application.

Figure 1

iOS propose de remplacer la barre d'actions par le champ de saisie de la recherche (Fig. 1).

Ce champ est souvent caché au-delà du haut d'une liste. Il n'apparaît que si l'utilisateur déplace la liste vers le bas.

Figure 2

Une icône normalisée peut également être proposée dans une barre d'onglets.

Figure 3

Les recherches peuvent être intégrées à la recherche globale du système. Cela est accessible à gauche de la page principale de l'iPhone. Ou bien, deux ou trois clics sur le *home* permettent de déclencher une recherche globale.

Android propose d'ajouter une icône dans la barre d'actions.

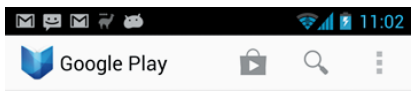


Figure 4

Le champ de saisie superpose la barre d'actions, tout en maintenant le menu. Des icônes peuvent alors disparaître pour être remplacées par des items du menu.

Figure 5

Les applications peuvent s'intégrer dans la recherche globale du smartphone.

Les anciens téléphones Android proposent parfois un bouton physique pour démarrer une recherche. Ce dernier disparaît, car les boutons système ne proposent pas d'équivalent. Les applications ne doivent pas exploiter ce bouton pour une recherche dans l'application.

Windows Phone propose un bouton physique de recherche globale qui n'est pas spécifique à l'application en cours. Ce bouton déclenche une recherche via Bing et non sur tout le périphérique. Il est nécessaire de proposer un bouton dans la barre d'actions.

Figure 6

La recherche globale n'est pas disponible au moment de la rédaction de ce document. Il semble qu'une prochaine version l'intégrera.

1.2 Ouvrir avec...

Des objets ou des documents peuvent être édités ou consultés par d'autres applications présentes sur le smartphone ou la tablette.

Les contraintes techniques des OS limitent parfois les possibilités offertes aux applications.

iOS permet l'ouverture d'un document par une autre application. Mais pour cela, ce dernier doit être recopié dans l'espace mémoire de l'application cible. Le document peut être présenté à l'utilisateur, mais il ne peut être modifié par une autre application. Cela limite les possibilités des conteneurs de fichiers dans le Cloud comme Dropbox.

Android permet un partage complet des fichiers entre les applications. L'utilisateur peut alors choisir l'application la plus à même d'effectuer des modifications. Par exemple, il existe de nombreuses applications capables d'améliorer une photographie. Il est possible d'éditer directement un fichier Dropbox.

Windows Phone ne permet pas l'édition de documents entre les applications, en dehors du Cloud. Un fichier peut être dupliqué dans le contexte d'une autre application pour être consulté.

Lors de la consultation d'un document sur SkyDrive (la solution Cloud de MS), la référence du document est utilisée par l'application d'édition pour lui permettre de modifier le fichier présent dans les nuages. Cette modification est alors visible par les autres applications utilisant SkyDrive. En cas de perte de réseau, impossible de modifier un fichier d'une autre application.

1.3 Partage avec...

Les applications veulent permettre le partage d'informations vers les réseaux sociaux ou autres applications. Chaque OS intègre des approches différentes pour permettre cela.

iOS propose un bouton spécifique pour permettre le partage d'un document. Par défaut, une liste finie d'applications natives est intégrée dans l'application. Cependant, il est possible de concevoir son application de façon à ce qu'elle puisse partager des fichiers avec certaines applications tierces.

Figure 7

Android propose une icône spécifique à ajouter dans la barre d'actions pour le partage. La liste des applications prêtes à recevoir un partage est variable. En effet, une application peut s'enregistrer comme cible d'un partage. Le dernier partage effectué est mémorisé, permettant à l'utilisateur de partager à nouveau plus rapidement. Dans l'exemple suivant, le Bluetooth est le dernier partage utilisé.

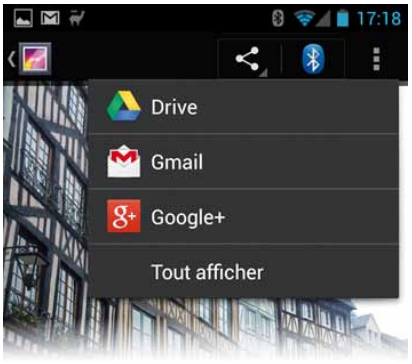


Figure 8

Windows Phone propose également un partage avec les différentes applications présentes dans le smartphone. La barre d'actions doit proposer un menu pour cela. Cela ouvre une nouvelle fenêtre pour sélectionner l'application cible. Elle est automatiquement générée par le système.

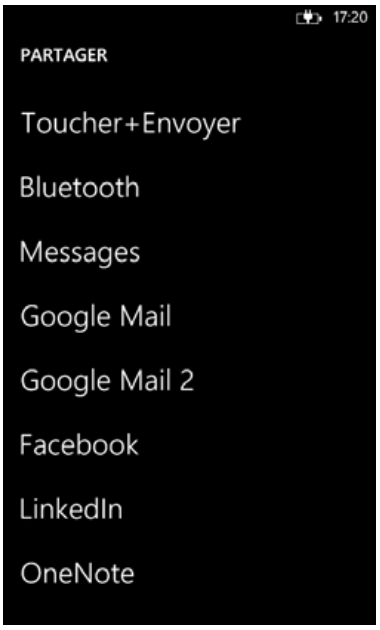


Figure 9

Analyse

Pour être intégrée au mieux, une application doit envisager d'être la cible ou la source d'un partage.

Sous iOS, l'application doit intégrer une liste finie d'API pour chaque réseau social, si elle ne souhaite pas être limitée aux propositions de l'OS.

1.4.Sélection multiple

Les utilisateurs désirent parfois effectuer une même action sur différents objets. Par exemple, ils désirent sélectionner certaines photographies pour les partager, les envoyer par mail ou les effacer.

iOS propose de passer par le bouton ***modifier***. Le mode de sélection multiple est alors indiqué dans la barre de titre.

Des cases à cocher sont superposées sur les photos.

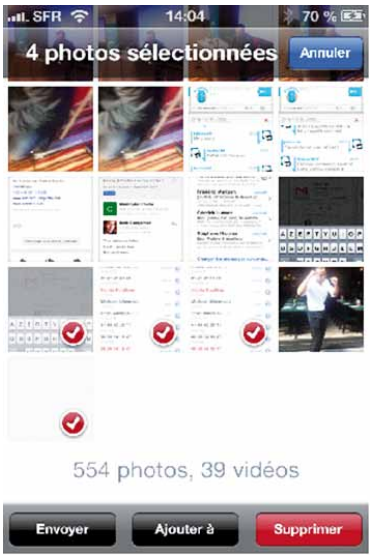


Figure 10

Android propose d'entrer dans le mode de sélection par un appui long sur un des éléments (grammaire invisible). La barre d'actions est alors modifiée pour proposer différentes actions à l'utilisateur. Il peut valider ou abandonner la sélection. Les éléments de l'écran peuvent également s'enrichir d'une coche ou d'un autre signal visuel de sélection. Le menu correspondant s'adapte à la sélection courante.

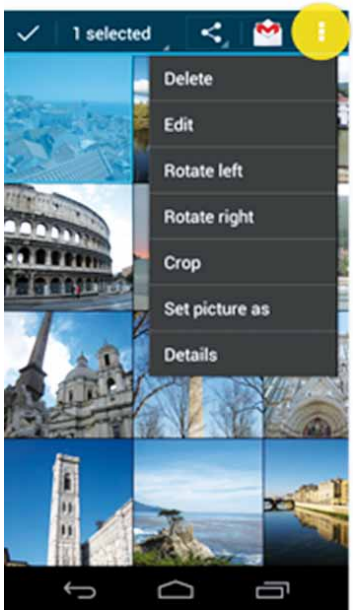


Figure 11

Figure 12

Les versions précédentes d'Android proposaient une action dans le menu pour entrer dans ce mode.

Windows Phone propose d'entrer dans le mode de sélection multiple à partir d'une action de la barre de menu.

Figure 13

Analyse

iOS et Windows Phone indiquent clairement à l'utilisateur qu'il peut entrer dans le mode de sélection multiple.

Android s'appuie sur une gesture, que l'utilisateur risque de ne pas connaître.

2 Paramétrage

Les OS utilisent des approches différentes pour la gestion des paramètres des applications.

iOS propose de centraliser toutes les options de paramétrage à un même et unique endroit. Ce qui a pour avantage de limiter le nombre d'éléments et d'options au sein d'une application. Par contre, cela oblige l'utilisateur à sortir de son application pour changer un paramètre (grammaire invisible). Il est probable qu'un utilisateur ignore la présence de paramètres pour son application.

Figure 14

L'utilisateur n'a pas à valider ses modifications pour qu'elles soient prises en compte. Elles sont effectives immédiatement.

Figure 15

Android propose d'utiliser un bouton d'action pour entrer dans le paramétrage. Une feuille de styles normalisée doit être utilisée pour valoriser les différents paramètres. Le modèle consiste à permettre la modification des paramètres sans demander de validation à l'utilisateur.

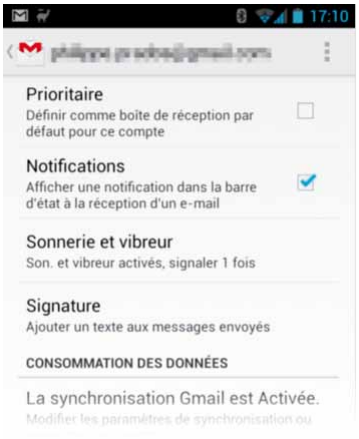


Figure 16

Windows Phone propose également d'ajouter un bouton d'action ou un menu pour le paramétrage. L'utilisateur ne doit généralement pas valider ses modifications, même s'il existe des applications qui proposent le contraire (Mail).

Figure 17

3 Panorama

Les écrans des smartphones sont de petite taille. Il est souvent nécessaire d'exploiter plusieurs vues mises les unes à côté des autres. Cela permet de proposer un écran système plus grand que l'écran physique et de présenter un extrait du panorama. L'utilisateur doit glisser l'écran à droite ou à gauche pour révéler la suite du panorama.

L'utilisateur n'a pas forcément conscience de la présence des autres vues du panorama. Plusieurs approches ont été imaginées.

iOS propose de signaler cela à l'utilisateur en présentant un indicateur de page (*page control*). Ce composant indique le nombre de pages disponibles et la page en cours de visualisation (grammaire visible). Le point plus clair indique la position relative de la page dans le panorama. Une icône spécifique peut remplacer le point.

Figure 18

Android propose le concept de page à glisser, mais n'offre pas de composant graphique équivalent au « page control ». Chaque application décide alors de le proposer ou non (avec des points, des barres, etc.). Par exemple, la page du bureau colorie pendant un instant un segment de la barre séparant les icônes du menu. Il existe des librairies pour proposer différents indicateurs de page.

Un débordement peut être volontairement utilisé pour signaler à l'utilisateur qu'il ne voit pas l'intégralité des informations. C'est le cas de la galerie par exemple.

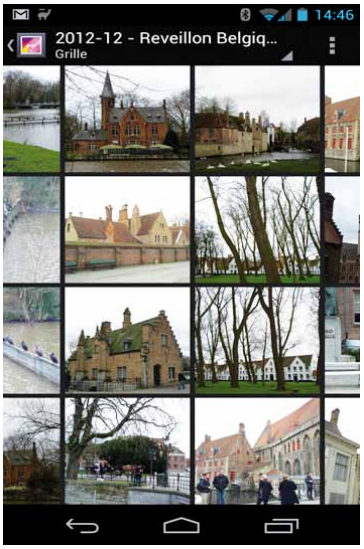


Figure 19

Dans l'exemple suivant, la taille des images est adaptée dès qu'une image ne tient plus à l'écran, pour être certain de générer un débordement (Fig. 19).

Windows Phone considère comme important que l'utilisateur ait conscience qu'il est possible de consulter les écrans complémentaires (les vues du panneau). Pour l'informer, plusieurs stratégies sont utilisées.

La première approche consiste à réserver une bande d'un demi centimètre à droite pour y présenter le bord gauche de l'écran suivant.

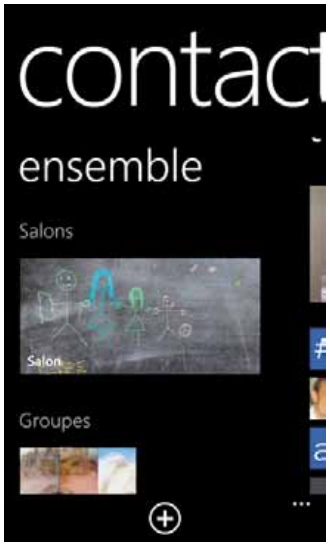


Figure 20

La deuxième approche consiste à utiliser un titre avec une police de grande taille dont le texte déborde en largeur de l'écran. Ainsi, l'utilisateur devine qu'il est possible d'aller voir à droite. Il est à noter que ce mode n'est pas disponible pour une présentation en mode paysage.

Certaines langues ne permettent pas de présenter l'intégralité du titre sur l'ensemble des volets, d'autres utilisent des mots trop courts pour que ceux-ci débordent.

Dans l'exemple suivant en français, impossible de connaître l'intégralité du titre, même si on le devine, car il n'y a pas de vue à droite.

Figure 21

Analyse

Le nombre de pages doit être limité lors de l'utilisation d'un panorama. Il est important de prévoir une place pour l'indicateur de page.

L'approche de Windows Phone réduit la place disponible aux données essentielles des applications. La traduction des applications Windows Phone est plus délicate, car elle a un impact direct sur l'ergonomie.

4 La connexion avec le réseau

Les applications exploitent fortement le réseau afin d'étendre les capacités du terminal. Les données peuvent arriver à la demande ou en tâche de fond. Cela peut déclencher des alertes ou des notifications à l'utilisateur.

4.1 Rafraîchissement à la demande

Les informations récupérées sur Internet sont une image à un instant donné, présente sur le serveur. Entretemps, de nouvelles données peuvent arriver, sans que l'application en soit prévenue. Par exemple, de nouveaux messages peuvent être publiés sur les réseaux sociaux.

Pour rafraîchir l'application, il est possible que le serveur informe directement l'application, mais plus souvent, l'utilisateur doit effectuer une action pour demander le rafraîchissement. Les OS proposent des approches différentes pour répondre à ce besoin.

iOS propose le pattern « Tirer pour rafraîchir » (*pull-to-refresh* – brevet de Twitter). L'utilisateur doit tirer au-delà d'une liste vers le bas, pour la faire déborder et déclencher une requête réseau pour obtenir les nouveaux éléments. Mais on peut aussi trouver un bouton permettant de demander un rafraîchissement. Il est à noter que le fait de tirer vers le bas est également utilisé pour faire apparaître une boîte de recherche (gesture).

Android propose quant à lui d'ajouter un bouton comme dernier élément d'une liste ou de présenter un sablier dès que l'utilisateur dépasse le dernier élément. Un bouton de rafraîchissement est également proposé. On ne retrouve jamais d'élément caché au-dessus de la liste. Le design et les animations des listes ne s'y prêtent pas.

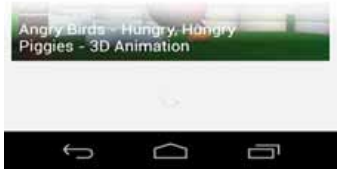


Figure 22

Windows Phone propose des listes infinies avec chargement des données au fur et à mesure. En cas d'attente, un loader indique que les données arrivent. Un bouton standard permet plus généralement de rafraîchir les données.

4.2 Signaler un événement

Les applications effectuent des traitements en tâche de fond. Elles peuvent parfois avoir des difficultés à fonctionner (perte de réseau, SMS qui ne part pas, mot de passe devenu invalide, etc.) Il est nécessaire de prévoir un mécanisme ergonomique pour signaler ces situations à l'utilisateur.

Plusieurs approches sont disponibles. Partant de l'idée que l'utilisateur est devant son écran lors de la diffusion de l'information, il est possible d'afficher une alerte par-dessus l'écran actif. Suivant les OS, ce message peut venir de l'application active ou d'une autre application exécutée en tâche de fond.

Cette approche présente plusieurs inconvénients : l'utilisateur risque d'être gêné par l'apparition de ce message, car il cache une partie de l'application. D'autre part, l'utilisateur peut manquer le message. Il n'est alors pas prévenu que ses mails ne sont plus téléchargés ou qu'un SMS n'est pas parti. Bien entendu, il ne faut informer l'utilisateur que si cela est vraiment important.

Pour simplement confirmer une action, il est préférable d'avoir un retour visuel sur l'écran, comme le changement de couleur d'un bouton.

Figure 23

iOS propose des notifications servant à alerter l'utilisateur qu'un événement

Figure 24

s'est produit. À tout moment, il est possible de consulter un écran, appelé « centre de notifications ». On y accède en glissant le doigt du haut vers le bas à partir de la barre de statut. Les alertes sont classées par applications (Fig. 24).

Il est possible d'être notifié de deux manières différentes : par une pop-up et une bannière. Un badge sur l'icône de l'application peut également signaler l'arrivée d'un événement.

Figure 25

Les notifications sur iOS sont standardisées et ne peuvent être personnalisées.

Figure 26

Elles indiquent le nom de l'application ainsi qu'un message. Il est possible de faire les actions suivantes : fermer la notification, ouvrir l'application concernée ou demander un rappel.

Un appui sur une alerte lance l'application correspondante.

Les notifications ne peuvent s'agréger en une seule dans la barre de notification.

L'application n'est pas prévenue de l'arrivée d'un événement. Une fois lancée, elle peut consulter ces événements pour réagir en conséquence et par exemple mettre à zéro le badge de l'icône.

Android propose quatre approches pour informer l'utilisateur : un texte défilant sur la barre de statut, des alertes par-dessus l'application, des notifications et des boîtes de dialogue pop-up.

Le développeur peut modifier les styles des alertes. Ces messages peuvent être placés n'importe où sur l'écran, mais ne sont pas actifs. Ils peuvent apparaître à l'initiative d'une autre application que celle actuellement présente à l'écran. L'utilisateur ne peut faire disparaître cette fenêtre manuellement, mais elle disparaît après quelques secondes.

Figure 27

Il est également possible d'afficher une boîte de dialogue à l'utilisateur pour lui demander de prendre une décision. C'est le cas, par exemple, lors d'une association Bluetooth. Ces dialogues peuvent être à l'initiative d'autres applications et surgir à tout moment. La transparence

Figure 28

autour du dialogue n'est pas complète. Il n'y a aucune limitation sur le contenu du dialogue (Fig. 28).

Android propose également une liste de notifications. Un glissé du doigt depuis le haut de l'écran permet à l'utilisateur de les consulter.

Figure 29

L'utilisateur peut glisser chaque événement à l'extérieur de l'écran, sur la droite ou la gauche, pour le supprimer.

Pour signaler la présence de notifications, des icônes spécifiques peuvent être ajoutées dans la barre de statut, présente sur tous les écrans. Ainsi, l'utilisateur est invité à consulter les notifications si une icône spécifique l'intéresse. Il n'a pas besoin de consulter régulièrement la liste pour savoir s'il s'y passe quelque chose d'important.

Figure 30

Un texte peut s'afficher sur la barre de statut lors de l'événement.

Les notifications peuvent être plus ou moins riches suivant les versions de l'OS.

Figure 31

Si elles ont besoin de plus de place, elles peuvent s'étendre directement depuis la liste des notifications.

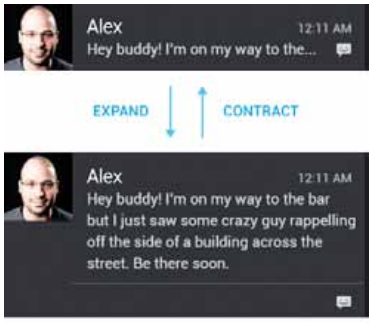


Figure 32

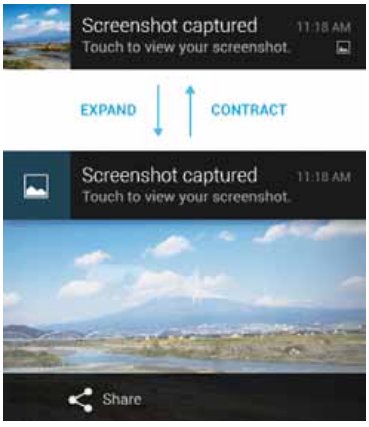


Figure 33

Elles peuvent proposer des actions simples, portées par des boutons.

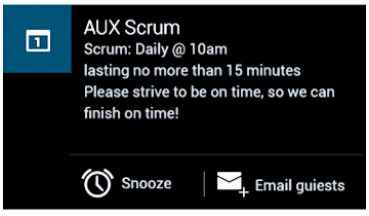


Figure 34

Pour éviter de polluer cette liste, les applications doivent les agréger au maximum. Si le même message arrive plusieurs fois, un compteur doit être incrémenté sur le premier message.

Android propose sur certains périphériques une LED, avec différentes couleurs, qui permet d'informer l'utilisateur sans que ce dernier ait besoin d'allumer son smartphone. La présence de cette lumière et les couleurs disponibles sont variables suivant les périphériques.

Les anciennes versions ne proposent que l'affichage d'éléments simples dans le centre de notifications. Il est possible d'avoir deux lignes de texte et une icône.

Windows Phone ne propose pas de conteneur centralisé de notifications. Le système propose quatre approches pour notifier l'utilisateur :

- Utiliser un label à la place de la barre de statut tout en haut lorsque l'utilisateur démarre l'application. Lors du lancement de l'application, cette dernière affiche furtivement une synthèse des actions passées ;

Figure 35

- Ajouter un compteur sur la tuile de l'application ; l'application doit proposer un paramètre pour ne plus mettre à jour la tuile en tâche de fond ;

Figure 36

- Afficher une alerte par-dessus l'application active. Un clic sur l'alerte lance l'application correspondante. Les applications peuvent refuser d'être ainsi perturbées ;

Figure 37

- Utiliser une boîte de dialogue qui surgit sur l'application, si elle est active. Le reste de l'espace est estompé.

Figure 38

Les applications ne peuvent pas signaler un problème à l'utilisateur s'il n'est pas présent devant l'écran. Tant que l'utilisateur ne relance pas l'application, il ne sait pas s'il y a eu un problème. Il est alors possible que ce dernier pense

Analyse

Les approches des différents OS sont très différentes. Les concepteurs doivent en tenir compte pour concevoir les applications.

Un utilisateur Android aura plus de latitude pour réagir, car les notifications sont actives, nombreuses et souples.

Un utilisateur Windows Phone devra installer une tuile s'il ne souhaite pas louper une alerte.

Un utilisateur iPhone sera libre de sélectionner le mode de notification globale que peut exploiter une application.

Les alertes lancent l'application correspondante sous iOS et Windows Phone, mais pas sous Android.

La présence d'une lumière est un excellent moyen d'informer l'utilisateur sans l'obliger à allumer régulièrement son smartphone.

que ses mails sont récupérés et qu'il n'y en a pas de nouveau, alors qu'en réalité, son mot de passe a expiré et qu'il ne l'a pas mis à jour dans son téléphone.

Les applications doivent demander à l'utilisateur l'autorisation de lui envoyer des notifications.

Il n'existe pas de lumière pour signaler un événement sans allumer le téléphone.

4.3 Widget actif

Les notifications permettent d'informer l'utilisateur, mais imposent que ce dernier utilise la liste pour les consulter. Il est également possible d'informer l'utilisateur d'informations concernant ses applications, en enrichissant dynamiquement un widget spécifique. Il s'agit d'une sorte de mini-application disponible sur le bureau.

Si l'utilisateur estime qu'une application est particulièrement digne d'intérêt, il peut décider d'installer le widget sur son bureau afin d'avoir une vue rapide de l'évolution de ses réseaux sociaux, de la météo, des informations, du trafic routier, etc.

Les approches sont assez différentes suivant les OS.

Pour **iOS**, la notion de widget n'existe pas. Il est simplement possible d'ajouter un indicateur numérique sur l'icône de l'application. Ce dernier permet d'informer de la présence d'un nouveau mail par exemple. Ce « badge » est valorisé par l'application ou par un événement venant du net.

Il existe une exception : l'application calendrier. L'icône de cette application a un affichage dynamique puisqu'elle présente une partie de la date du jour (nom du jour de la semaine + N°).

Figure 39

Android permet aux applications d'exposer des widgets de différentes formes et différentes tailles, plus ou moins actifs. Cela permet d'organiser les différents écrans composant le bureau, afin d'avoir soit des accès directs à certaines fonctionnalités des applications, soit des informations fraîches. Il est possible d'interagir directement depuis le bureau. Les widgets peuvent être simplement informatifs (la météo, un dossier de mail, l'agenda), avoir quelques boutons (activer le Wifi, le Bluetooth, itinéraire vers le domicile) ou une combinaison de boutons (contrôle de la playlist). L'utilisateur peut modifier la taille des widgets.

Figure 40

L'espace disponible sur un écran du bureau étant limité, il est souvent nécessaire de les organiser sur les différents écrans disponibles. Les widgets peuvent être présents sur l'écran de déverrouillage.

Windows Phone utilise des tuiles (widget applicatif) pour permettre à

Figure 41

l'utilisateur d'avoir un résumé rapide de l'état de plusieurs applications. Elles sont capables de se rafraîchir automatiquement et de lancer l'application. Comme sur Android, c'est à l'utilisateur de sélectionner et d'organiser les widgets présents sur son bureau. Cet écran ne peut être qu'en portrait (Fig. 41).

Contrairement à Android ou iOS, l'utilisateur dispose d'un espace infini vers le bas pour agréger tous les widgets. Un rapide glissement du doigt permet alors d'avoir une vue synthétique des informations. Par contre, les widgets ne peuvent que présenter de l'information. Aucune action n'est possible. Ils sont mis à jour via des notifications du réseau.

Les tuiles peuvent exister en trois tailles et différents modes d'animation : Flip (la face se retourne sur place), Icône (la tuile est fixe), Cycle (une succession d'images) ou Dynamique. Elles doivent avoir un fond coloré uni.

Figure 42

Un appui sur une tuile lance l'application correspondante. Il est impossible d'interagir plus finement. Par exemple, impossible de contrôler la diffusion musicale depuis le bureau. C'est le bouton « volume » qui fait apparaître un menu par-dessus l'application.

Figure 43

Il est également possible d'ajouter des informations sur l'écran de verrouillage, au choix de l'utilisateur. L'application peut livrer le texte sous la date et une icône parmi cinq avec un chiffre.

Figure 44

Analyse

Un utilisateur Android pourra agir directement depuis le bureau.

Un utilisateur Windows Phone sera à jour sur les données présentées.

Un utilisateur iOS devra se contenter d'un badge sur l'icône de l'application.

Plus vos widgets sont dynamiques, plus les utilisateurs vont se retourner sur vos applications.

4.4 Tâche de fond

Il arrive fréquemment qu'un utilisateur de mobile n'ait pas accès au réseau. Cela n'implique pas pour autant qu'il ne puisse pas réaliser d'action sur son téléphone. Il peut parfaitement rédiger un mail, réagir au flux d'un réseau social, publier une nouvelle photo, etc. Les données ne sont pas diffusées immédiatement, mais le seront dès que le réseau sera disponible, ou lors de la prochaine connexion Wifi.

De même, il est intéressant de récupérer des données dès que possible, afin d'alimenter les applications. Ainsi, même en l'absence de réseau, l'utilisateur peut consulter les derniers mails récupérés.

iOS ne permet pas de publier des modifications si l'application n'est pas active. Les données ne peuvent être diffusées sur le réseau. Tant que l'utilisateur ne relance pas l'application, les données restent en attente. Il n'est pas possible de récupérer des données en tâche de fond, sauf pour l'application mail native. Il n'est donc pas possible de consulter les derniers mails en cas de perte de connexion.

Les applications Apple ne subissent pas cette contrainte. Il y a donc une différence entre rédiger un mail sans réseau avec l'application d'Apple et la même action avec une solution alternative. C'est uniquement lors du réveil de l'application alternative que les messages pourront partir.

Quelques actions en tâche de fond sont possibles, comme la navigation GPS, l'écoute d'un flux audio ou la téléphonie IP.

Android permet la publication de services. Ce sont des traitements exécutés en tâche de fond ou sur événement. Ainsi, il est possible de publier les messages en attente, dès la reprise du réseau. L'utilisateur peut être en train d'utiliser une autre application ou avoir le téléphone éteint, les messages seront envoyés dès que possible. Il est possible de récupérer les nouveaux mails, au fur et à mesure qu'ils arrivent. Ainsi, même en absence de réseau, il est possible de les consulter, alors que l'application n'a jamais été lancée par l'utilisateur.

Window Phone propose une approche de multitâche sous le contrôle de l'OS. Périodiquement, à l'initiative du système, des traitements peuvent être déclenchés. Ces derniers peuvent envoyer ou recevoir des données. Si une application n'est pas lancée pendant un certain temps, les tâches de fond associées sont considérées comme inutiles. Les traitements ne sont alors plus déclenchés.

Il existe également des traitements déclarés comme gros consommateurs de ressources. Ils ne sont déclenchés par le système qu'avec une connexion Wifi et un branchement au secteur.

Analyse

Les approches d'Android et de Windows Phone sont plus tolérantes à la perte du réseau.

iOS ne permet pas de pré-charger des données pour les exposer à l'utilisateur, sans réseau.

5 Présentation à l'écran

L'écran d'un smartphone présente un espace réduit qu'il faut exploiter au maximum. Il doit également permettre de présenter un retour immédiat à l'utilisateur (affichage de la lettre du clavier, modification de la couleur d'un élément, etc.). De plus, le terminal peut basculer en mode portrait ou paysage et propose un clavier virtuel.

5.1 Portrait/paysage

Les applications peuvent être utilisées sur téléphone ou sur tablette. L'usage principal sur un smartphone et le mode portrait. Pour une tablette, c'est le mode paysage qui est privilégié. Les applications devraient être compatibles avec ces deux modes.

iOS ne permet pas à toutes les applications de fonctionner dans les deux modes. Généralement, les applications spécifiques au téléphone n'acceptent que le mode portrait. Les versions pour iPad sont alors très différentes. Par exemple, la gestion des contacts entre iPhone et iPad est très différente.

Android ne propose que des applications qui fonctionnent dans les deux modes, même si les applications typiques téléphone sont présentées en mode portrait (appel téléphonique, SMS). En effet, certains téléphones possèdent un clavier physique qui force le mode paysage lors de son utilisation. Chaque saisie peut passer l'application en mode paysage.

Windows Phone ne propose pas le mode panorama avec une utilisation

Analyse

Nous préconisons de toujours prévoir les deux modes d'accès : portrait et paysage. Ainsi, la même application fonctionnera sur smartphone et tablette, avec ou sans clavier, et demain sur une télévision.

L'utilisation du panorama sur Windows Phone est difficile, car il ne fonctionne qu'en portrait.

CHOISISSEZ ~~ENTRE~~
LA TECHNOLOGIE & LE SERVICE

À PARTIR DE
19,99€ HT/MOIS

Processeur Intel dernière génération*

* à partir du G460

RAM jusqu'à 96 Go DDR3

Disque jusqu'à 8 To SATA ou 4 x 240 Go SSD
Disques additionnels jusqu'à 3 To

TECHNOLOGIE DE POINTE | DATACENTER PROPRIÉTAIRE | SERVICE CLIENT ★★★★

Support Technique
Nos équipes sont disponibles sur site en 24x7.

Gérez votre serveur sur une interface intuitive
Reboot, Reverse DNS, Rescue MX, Backup, RAZ...

Disponible sur votre Espace Client

RENDEZ-VOUS SUR : express.ikoula.com/serveur-dedie

01 84 01 02 50
sales@ikoula.com

ikoula Hosting Services

NOM DE DOMAINE | MESSAGERIE | HÉBERGEMENT | CERTIFICAT SSL | CLOUD | SERVEUR DÉDIÉ

en paysage. Donc beaucoup d'applications ne fonctionnent pas dans cette orientation.

5.2 Dialogue

Lors du besoin d'une intervention de l'utilisateur, une boîte de dialogue est proposée à l'utilisateur. Cette dernière propose généralement quelques boutons, une liste réduite de choix ou de cases à cocher. Plusieurs approches coexistent pour l'ordre des boutons.

iOS met l'action principale à droite.

Figure 45

Android propose que le bouton de validation soit le dernier.

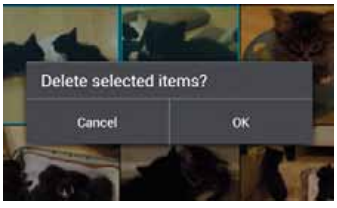


Figure 46

Les anciennes versions d'Android utilisaient un ordre inversé, avec la validation à gauche.

Windows Phone propose le bouton de validation à gauche.

Figure 47

Il est important de respecter ces conventions, car l'utilisateur a tendance à cliquer sans vraiment faire attention aux libellés des boutons.

Analyse

Nous préconisons de respecter le modèle de l'OS, même si nous préférons le bouton de validation à droite, en raison du fait que l'utilisateur va scanner les options de gauche à droite avant de prendre sa décision. Si l'option primaire est placée à droite, l'utilisateur n'aura pas besoin de faire d'effort de mémorisation ni de revenir sur les boutons précédents.

6 Sécurité

Chaque OS mobile propose son approche de la sécurité. Certains préfèrent exposer tous les privilèges avant l'installation de l'application, d'autres préfèrent demander l'accord de l'utilisateur au premier usage.

iOS propose un modèle de sécurité simplifié, où l'utilisateur doit accepter chaque privilège avant la première utilisation. Les privilèges sont à ce jour : GPS, Photo, Contacts, Calendrier, Rappels, Partage Bluetooth, Compte Twitter et Compte Facebook.

Figure 48

Android propose un modèle basé sur une exposition des privilèges à l'utilisateur avant l'installation de l'application. Comme le modèle de sécurité est complexe et évolutif, les nombreuses informations présentées ne permettent pas toujours aux utilisateurs d'avoir conscience des impacts. Ils ont tendance à accepter tous les privilèges (Fig. 49).

Il n'est pas possible de sélectionner les privilèges à accorder. Cela simplifie la conception des applications, car il n'est pas nécessaire de prévoir les cas où un privilège n'est pas disponible.

Figure 49

Des techniques avancées permettent d'ajouter dynamiquement de nouveaux privilèges. Cela est une sorte d'accord de l'utilisateur. Mais pour cela, l'utilisateur devra quitter l'application pour installer une extension via la place de marché.

Windows Phone propose d'indiquer l'ensemble des privilèges d'une application avant son installation. De même, il n'est pas possible de sélectionner certains privilèges ou d'en ajouter en cours de route.

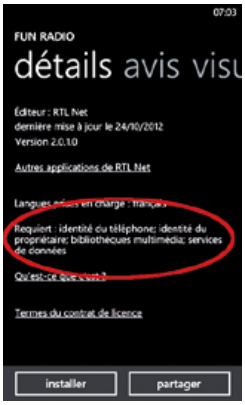


Figure 50

Conclusion

Nous avons identifié de nombreuses différences entre les OS et de nombreux composants à exploiter lors de la conception d'une application mobile. Nous n'avons pas la prétention d'être exhaustifs. Le tableau suivant synthétise en une vingtaine de points les différentes informations. ■

Type	iOS	Android	WP8
		Barre de statut	
Visible	Système	Notifications applicatives	
Invisible			Swipe vers le bas
		Titre de l'application	
Visible	Au centre	Avec icône de l'application ou de l'objet de la page.	Grand ou Petit
Invisible	Sauf si trop d'actions.		
		Navigation	
		Monter d'un niveau	
Visible	Bouton action, avec nom de l'écran parent.	Appui sur titre, chevron à gauche.	Appui sur titre, chevron à gauche.
Invisible			Bouton physique « retour ».
		Retour à l'écran précédant	
Visible	Bouton action, avec nom de l'écran précédant. Impossible entre applications.		
Invisible		Bouton système	Bouton physique
		Suivant/Précédent	
Visible	Boutons dans barre d'actions. Impossible entre applications.		Boutons dans barre d'actions.
Invisible		Swipe de l'écran	
		Les Onglets	
Visible	En bas de l'écran avec icône et texte, ordre modifiable par l'utilisateur.	Avec la barre d'actions, en dessous ou en combo-box. Ordre des onglets non modifiable.	Sous-titre (sans délimiteur). Ordre non modifiable.
		Les filtres	
Visible	Contrôle de segmentation	Onglets	Onglets
		Menu général (Drawer)	
Visible	Icône à gauche de la barre d'actions.	Flèche à gauche de l'icône sur barre d'actions.	Icône à gauche de la barre d'actions.
		Les actions de l'utilisateur	
		La Barre d'actions	
Visible	Limité par la navigation. Utiliser une autre barre.	Texte et/ou icône, sans limite du nombre d'actions. Peux être coupée en deux parties.	Icône et texte, sans limite du nombre d'actions
		Menu contextuel d'un objet	
Visible	Chevron à droite de l'item. Mène vers un autre écran.	Triangle de menu, local à l'écran. Si « ... » ouvre un sous-menu.	Pas de symbole. Ouvre un sous-menu.
Invisible	swipe	swipe	Appui long
		Recherche	
Visible	Dans la barre d'actions. Critère dans la barre d'actions.	Dans la barre d'actions. Critère dans la barre d'actions.	Dans la barre d'actions. Critère par-dessus le titre.
Invisible	Ou caché au-dessus d'une liste	Bouton physique, sous Android v2 (à bannir).	

Type	iOS	Android	WP8
		Ouvrir avec...	
Visible	Dans barre d'actions. L'application cible ne peut pas modifier le fichier.	Dans la barre d'actions. L'application cible peut modifier le fichier.	Dans la barre d'actions. L'application cible ne peut pas modifier le fichier.
		Partage avec...	
Visible	Dans barre d'actions, superpose fenêtre avec choix imposé.	Dans la barre d'actions, dépend des applications présentes, mémorisation du dernier choix.	Dans la barre d'actions, dépend des applications présentes.
		Sélection multiple	
Visible	Dans barre d'actions. Ajoute des cases à cocher ou des cadres.		Dans barre d'actions. Ajoute des cases à cocher ou des cadres.
Invisible		Appui long. Change le menu, ajoute des cases à cocher ou des cadres.	
		Paramétrage	
Visible		Dans l'application, sans validation	Dans l'application, sans validation
Invisible	En dehors de l'application, sans validation		
		Panorama	
Visible	Page contrôle, swipe	Page contrôle, swipe	Titre, Débord de la page suivante, swipe
		Connexion avec le réseau	
		Rafraîchissement à la demande	
Visible	Dans la barre d'actions	Dans la barre d'actions ou en dernier élément de la liste	Dans la barre d'actions ou en dernier élément de la liste
Invisible	Pousse vers le bas/haut	Arrive au dernier élément	
		Signaler un événement	
Visible	Alerte, dialogue, notification, badge sur logo, son	Texte roulant sur barre de statut, icône animée sur barre de statut, bannière, notification agrandissable avec actions, widget, son, LED, boite de dialogue	Bannière, widget, boite de dialogue. Demander l'accord de l'utilisateur dans l'application.
Invisible	vibration	vibration	vibration
		Widget	
Visible		Widget avec action sur le bureau	Tuile
		Tâche de fond	
Visible	Pendant l'application	Pendant l'application	Pendant l'application
Invisible		En tâche de fond	
		Présentation de l'écran	
		Portrait/paysage	
Invisible	En principe, pour toutes les applications	En principe, pour toutes les applications	Portrait si utilise le mode Pivot, sinon Portrait ou Paysage.
		Boite de dialogue	
Visible	Validation à droite	Validation à droite	Validation à gauche
		Sécurité	
Visible	Lors du premier usage		
Invisible		Lors de l'installation de l'application	Lors de l'installation de l'application

GREAT PLACE TO WORK Best Workplaces 2013 France

UNIVERSUM TOP 100 IDEAL EMPLOYER 2013 STUDENT SURVEY

IMAGINER ET CONCEVOIR UN MONDE SANS CONTRAINTE

Technologies de pointe, challenges complexes en systèmes embarqués, ambitions internationales. Avec Parrot, faites le pari de l'innovation dans l'univers des périphériques sans fil.

BOUSCULEZ TOUS LES CODES SUR
WWW.RECRUTE.PARROT.COM

UN MONDE D'INNOVATIONS AU CŒUR DE PARIS

Parrot®

TRAFIQUER UN MOTEUR :

C'EST SIMPLE COMME ÉLEVER UN ENFANT !

par P-e-G

Le but de cet article est d'intervenir dans le code d'un moteur de base de données (H2 pour ne pas le citer), afin d'ajouter deux fonctionnalités intéressantes : le « Row/Cell Level Security » (sécurité au niveau d'une ligne de table de base de données) et un accès distant aux données. Plus que ces fonctionnalités, cet article nous permettra d'étudier une base de données relationnelle, non pas comme d'habitude d'un point de vue « client » (JDBC, ODBC et consorts), mais de l'intérieur du moteur.

1 Cela arrive même aux meilleurs

```
List<BriquesLego> legos ;
For (BriquesLego lego : legos )
{Lego.metsUnAnimalDessus(petitChien)}
```

« *Paaapaaaaaaaa, il y a NullPointerException qui s'affiche !* »

C'est dans ces cris et ce morceau de code que j'ai commencé mon samedi matin. Ce code, que je n'ai pas écrit, est celui que mon petit Naël a élaboré afin de programmer un moteur de *Path Tracing* sous Windows 8. Malheureusement, son code ne marche pas et j'avoue que cela m'inquiète un peu. En effet, que Naël, qui n'a pas encore trois ans, se mette à la programmation objet ne me surprend guère : si vous avez lu mes précédents articles, vous savez comme moi que l'informatique n'a aucun secret pour lui depuis ses six mois. Mais que mon petit garçon fasse une erreur de code aussi simple me chiffonne quelque peu. C'est pourquoi, armé d'une tétine, d'un mouchoir, d'une peluche et d'un biscuit, je me suis engagé dans une discussion de père à fils avec lui : « *Alors Naël, que se passe-t-il ? Ton code ne fonctionne pas ? Peut-être est-ce ton pointeur legos qui n'est pas initialisé ?* » Pris en défaut, Naël préfère garder le silence et me défier du regard avant de me lancer quelques secondes plus tard : « *Papa, quand petit frère sera né, est-ce que je vais devoir partager ? Qu'est-ce qui va changer ? Est-ce que ma vie va autant changer que la tienne lorsque je suis né ?* »

Je touche enfin le cœur du problème de mon petit Naël : ce n'est pas, contrairement à ce que tu penses tout bas lecteur sans cœur, qu'un enfant de trois ans utilise Windows 8, mais l'arrivée prochaine d'un petit frère suscitant des angoisses assez légitimes dans la tête de mon petit bonhomme ! J'ai bien lu dans *Parents Geek Magazine* qu'il ne fallait pas forcément tout dire à un enfant, et en particulier ce qui a changé pour moi avec sa naissance. D'autant plus que dans ma « vie d'avant », mes priorités allaient des filles à la drogue, en passant par le *coding* et le business...

Pas question donc de lui confier ce qu'était véritablement ma « vie d'avant », quand bien même il s'agit de Naël, celui pour qui le C n'est qu'une surcouche mal foutue à l'assembleur !

Je lui réponds alors par une pirouette dont j'ai le secret :

« *Ben Naël, avant ta naissance j'écrivais surtout des codes MOLAP et ce qui a changé, c'est surtout que je n'ai pas eu le temps de finir un article pour Linux Magazine sur le hacking des storage engines de DBMS.*

- Ahhh...Ok Papa, mais c'est quoi ce hack ? »

Je lui explique donc que j'avais dans mon « placard à articles non terminés » un papier inachevé sur MySQL et son hacking du *storage engine* dont le but est d'ajouter de la sécurité et de l'accès distant via XmlRPC.

« *Ben Papa, il faut le dire à Naël quand tu n'arrives pas à finir un article. Je vais au pot et quand Naël revient, il va t'aider à finir !* »

Tout gonflé de testostérone et de virilité, galvanisé par la façon dont j'avais su répondre à une question difficile (en évitant justement de répondre !), je décide alors de finaliser cet article inachevé.

2 Fonctionnalités d'accès distant aux données et de Row/Cell Level Security

Les moteurs de bases de données relationnelles font partie des outils informatiques qui m'intéressent le plus. Ceux-ci peuvent en effet posséder une double fonctionnalité majeure : être accessibles pour la plupart d'entre eux à travers un réseau (ce qui autorise la création d'applications distribuées) et conserver de façon efficace des giga, voire des téraoctets de données (nos applications distribuées peuvent donc travailler sur beaucoup des données).

Afin de pouvoir acquérir cette double fonctionnalité, les moteurs de bases de données s'équipent de nombreuses micro-fonctionnalités telles que l'indexation de données, l'optimisation du requêtage, ou encore la mise en place de la sécurité vis-à-vis des utilisateurs.

Néanmoins, il y a deux autres fonctionnalités qu'il me semble souhaitable de pouvoir ajouter dans ces moteurs : l'accès distant aux données et le Row/Cell Level Security **[1,2]**.

Commençons par la fonctionnalité la plus simple, qui est l'accès distant aux données. La plupart des moteurs de bases de données vivent sur un emplacement réseau et se situent au même endroit que leurs données (exemple : les données de MySQL sont là où se situe le moteur de MySQL). Par conséquent, il est difficile de désolidariser le moteur et les données. Mais pourquoi voudrait-on effectuer cette désolidarisation ?

Il y a au moins deux cas d'usage. Le premier est celui de la réactivité. En effet, lorsque le nombre d'utilisateurs est important, il est possible que le « bottleneck » de performance provienne du moteur de base de données. Ce cas se produit lorsque l'application cliente provoque de nombreuses requêtes SQL différentes. Ce large spectre de requêtes peut remplir le cache de requêtes préparé du moteur de base, ce qui provoque une sur-activité de la couche d'optimisation du moteur...et un écroulement des performances générales. Il est dans ce cas précis assez intéressant de déplacer le moteur de base de données vers un emplacement, non pas centralisé, mais sur chaque poste client. Autrement dit, les données restent stockées et centralisées sur un serveur de données et chaque instance de l'application cliente embarque un moteur de bases de données.

L'autre cas d'usage est celui des postes mobiles. Un poste mobile (portable, périphérique, tablette...) doit pouvoir utiliser certaines applications « clientes » lourdes de « partout ». Il est donc assez difficile de permettre à ces applications d'utiliser une base de données distante « en direct », les protocoles d'accès aux bases ne passant pas sur Internet. Une solution possible est d'encapsuler ces protocoles de bases de données dans du HTTP. Une autre solution est d'embarquer le moteur de base de données dans le poste mobile et d'autoriser ce moteur à accéder aux données sur Internet (via un protocole WebDAV, FTP et consorts).

Pour obtenir cette application sans trop débourser, il est possible de « monter » un disque réseau distant dans les postes clients et de signaler aux moteurs de bases de données qu'il faut stocker/lire/modifier les données sur ce disque réseau. L'autre possibilité, et c'est l'un des objectifs de cet article, est de modifier le moteur de base de données pour pouvoir accéder de façon transparente à un serveur de données.

L'autre fonctionnalité que nous allons mettre en place est le Row/Cell Level Security. Pour comprendre cette fonctionnalité, il faut savoir qu'un moteur de bases de données est équipé d'une sécurité permettant de filtrer la lecture/écriture des contenus et de la structure d'une base de données. Autrement dit, pour pouvoir lire les données d'une base, il faut être habilité à se connecter au moteur de base de données, que cette habilitation nous autorise à accéder à la base de données et enfin, que cette dernière nous autorise à lire/modifier les tables de cette base. Néanmoins, cette sécurité peut être considérée comme un « tout ou rien » : ainsi, soit un utilisateur a une autorisation sur la table d'une base (il peut lire, voire modifier des données), soit il ne peut rien faire dessus.

Supposons maintenant une table nommé « Personne » modélisant les employés d'une entreprise. Il est vraisemblable que cette table puisse contenir les noms, prénoms, numéros de bureau, ainsi que les numéros de téléphone de ces employés.

Supposons maintenant deux groupes pouvant accéder à cette base : le premier groupe « RH » représente les ressources humaines et le deuxième groupe « Intendance » modélise les employés faisant l'intendance des bureaux.

Il est assez difficile de définir que les personnes du groupe RH ne peuvent avoir accès qu'aux colonnes nom, prénom et numéro de téléphone et que les personnes du groupe Intendance ne peuvent accéder qu'aux colonnes nom, prénom et numéro de bureau.

Il est encore plus difficile d'implémenter un groupe d'employés « Sécurité » pouvant avoir accès à tous les numéros de téléphone des employés, sauf ceux commençant par un 06 (numéro de téléphone portable). En effet, il est légitime que la sécurité puisse appeler un employé sur son téléphone, mais si celui-ci est un téléphone portable, il s'agit peut-être d'un numéro privé ne pouvant être utilisé que par des personnes du groupe RH.

Dans ce cas, la solution classique et propre est de découper de façon assez complexe la table modélisant les employés en plusieurs sous-tables et d'affecter sur ce découpage un ensemble de droits RH, Intendance et Sécurité.

L'autre solution, moins propre, serait de laisser cet aspect de sécurité à la couche applicative. Il s'agit dans ce cas d'un pis-aller, car d'une part, la

sécurité des données doit se trouver naturellement dans la base de données et non dans un acteur tiers (l'applicatif) et d'autre part, rien ne dit que la couche applicative n'ait pas buggé.

Dernière possibilité : ajouter à la table de notre base de données des méta-informations explicitant la sécurité que nous voulons appliquer sur les données.

3 Architecture « simpliste » d'un moteur de base de données

J'espère que le lecteur, en découvrant les fonctionnalités que nous voulons ajouter, se doute que nous allons faire un hack dans le moteur de base de données lui-même. Afin de pouvoir y arriver, il est de bon ton de savoir grosso modo comment marche une base de données.

Donc, par avance, mes excuses aux lecteurs qui seraient déjà familiarisés avec les cœurs de bases de données (et qui peuvent donc sauter ce chapitre).

Il est possible de découper un moteur de bases de données en trois gros blocs. Le premier consiste à lire une requête (SQL pour les moteurs relationnels) et à la transformer en une structure interne du moteur. Le deuxième bloc intervient lorsqu'avec la requête ainsi modélisée, le moteur entame une tâche de planification/exécution de la requête. L'idée ici est de prendre en considération les tables touchées par la requête et de regarder comment exécuter la requête. En effet, il est assez trivial de voir que, pour une même requête SQL, il y a plusieurs façons de mener à bien cette requête. Autrement dit, comme le SQL est un langage qui décrit ce que l'on veut sans signifier comment on le veut, la tâche de planification/exécution a pour but de trouver un moyen de résoudre la requête.

Lorsque le moteur sait comment exécuter sa requête, ce dernier entame le troisième bloc moteur. Il va commencer à interroger la couche de *storage* des données. Cette couche a pour double objectif d'exposer la structure et les données des tables, ainsi que de sauvegarder de façon efficace les données sur le disque.

A ce titre, il est intéressant de lire les parties « virtual table » (exposition de structure de table) et « file format » (comment sont stockées les données sur le disque) de SQLite **[3,4]**.

C'est ici que les deux fonctionnalités dont nous avons parlé dans le précédent paragraphe vont se situer : le Row/Cell Level Security va en effet se positionner comme des tables virtuelles servant de proxy à de réelles tables et l'accès distant aux données tentera de simuler pour le moteur de bases de données un *file system* l'autorisant à lire/écrire ces tables.

4 H2

Afin d'être honnête avec le lecteur, je dois reconnaître avoir élaboré une première ébauche de cet article en me basant sur le moteur de MySQL et en mettant l'accent sur le fait que ce dernier supporte les *storage engines* sous forme de plugin. La première version de cet article n'est finalement jamais sortie, parce qu'il m'a été impossible d'écrire un papier « grand public » en prenant comme base un code aussi complexe que celui de MySQL. Ce papier a donc fini dans mon « placard à articles non terminés » jusqu'à ce que je tombe sur le moteur H2.

H2 est un moteur écrit en Java sorti en 2005. Il fait partie de la famille des moteurs accessibles, soit par le réseau, soit « embarquables » dans une application. Ce moteur possède entre autres deux atouts : le premier est sa taille relativement petite l'autorisant à être embarqué par exemple dans un téléphone Android, le second est d'être extrêmement bien écrit. Il prévoit par ailleurs plusieurs points d'extension qui nous intéressent ici, à savoir le fait de pouvoir définir des moteurs de gestion de tables personnelles (ce qui est quasi obligatoire pour le Row/Cell Level Security) et de pouvoir ajouter des systèmes de stockage.

Nos fonctionnalités peuvent ainsi non seulement s'intégrer facilement avec H2, mais aussi s'ajouter assez « facilement » dans MySQL (via le *plugable storage engine*), SQLite et PostgreSQL.

5 Accès distant aux données

Pour que cette fonctionnalité atteigne son objectif, et ainsi offrir un accès distant aux données, il faut d'abord savoir si d'une part, on veut un système en lecture/écriture ou simplement en lecture et d'autre part, quel type de protocole utiliser pour fournir cet accès

La différence entre un système lecture/écriture et un système simplement en lecture est de taille. La première contrainte est de pouvoir faire une lecture/écriture de façon « aléatoire » dans les fichiers. La notion d'aléatoire se comprend ainsi :

- Lorsque le moteur de bases de données lit les données, il utilise une structure particulière appelée « index ».
- L'index organise les données de façon à retrouver directement un enregistrement. Autrement dit, lorsque l'on demande la ligne 3 d'une table, l'index aide le moteur à retrouver directement la troisième ligne (ce qui évite la méthode de parcours de la table en passant par la ligne 1, la ligne 2 et enfin la ligne 3).
- Encore faut-il être capable de lire l'index (ce qui est très facile), puis de lire directement la troisième ligne (ce qui est bien plus difficile).

Faire un file system en lecture seule est donc relativement simple parce que l'on peut, « au pire du pire », simuler un accès en lecture aléatoire, ce qui donne la situation suivante :

- soit le protocole fournit la fonctionnalité d'accès en lecture aléatoire,

- soit l'on rapatrie la totalité du fichier en local dans un emplacement temporaire et on l'ouvre en mode lecture aléatoire.

L'écriture en mode aléatoire s'avère beaucoup plus difficile si le protocole sous-jacent ne l'autorise pas. Si ce dernier ne le supporte pas, il faudrait, pour simuler l'écriture en mode aléatoire, récupérer le fichier, le modifier et le rapatrier sur le serveur... sauf s'il a été modifié entre-temps. Il faut donc pourvoir locker le fichier à distance, le télécharger, le modifier et le remettre sur le serveur puis libérer le lock.

Or, la mise en place d'un lock nécessite une opération atomique, ce qui, sur un file system distant, est plus que complexe à mettre en œuvre.

Cette réflexion sur le mode en lecture/écriture aléatoire d'un fichier distant rejoint ainsi la question du protocole utilisé pour notre accès distant aux données.

A ce titre, nous pouvons regarder le petit tableau des différents protocoles suivants :

Protocole	Lecture aléatoire	Lecture/Écriture aléatoire
FTP	Oui	Non
HTTP	Oui	Non
WebDAV	Oui	Non
CIFS (Samba)	Oui	Oui

Dans cet article, nous allons utiliser la couche WebDAV pour créer notre module, parce qu'il s'agit d'un protocole Internet a priori non filtré (au contraire de CIFS) et que contrairement à FTP et HTTP, c'est un protocole « chic ». Essayez en soirée de parler de votre module d'accès FTP, vous ne ferez pas vibrer les foules, alors qu'avec le WebDAV c'est le succès garanti !

5.1 WebDAV, configuration sous Apache

Nous allons donc monter un serveur WebDAV de test. Pour mémoire, WebDAV est une extension du HTTP permettant d'exporter sur Internet un système de fichiers.

Après avoir configuré WebDAV, nous pouvons avoir accès avec le navigateur Internet au système de fichiers en question, mais il est aussi possible de monter ce dernier sous Linux ou Windows.

Le but de ce chapitre est de configurer rapidement un serveur WebDAV avec le serveur HTTP de la fondation Apache. L'idée est d'obtenir quelque chose de fonctionnel rapidement. Pour illustrer notre propos, nous avons choisi un serveur version 2.0. Il faut donc en premier lieu charger les bons modules WebDAV.

Pour ce faire, il faut éditer le fichier **`httpd.conf`** et ajouter ces deux lignes :

```
LoadModule dav_module modules/mod_dav.so
LoadModule dav_fs_module modules/mod_dav_fs.so
```

Il faut dire ensuite au serveur Apache quel répertoire local exposer sur Internet et sur quelle URL.

En ajoutant ces quelques lignes, le serveur WebDAV doit marcher :

```
DavLockDB "/tmp/DAVLOCK"
Alias /webdav "/tmp/ "
 <Directory "/tmp">
  Dav On
 Order Allow,Deny
    Allow from all
    Options Indexes
 BrowserMatch "^Jakarta-Commons-VFS" redirect-carefully
BrowserMatch "^Jakarta Commons-HttpClient/3.0" redirect-carefully
BrowserMatch "^Jakarta Commons-HttpClient/3.1" redirect-carefully
BrowserMatch "Microsoft Data Access Internet Publishing Provider"
redirect-carefully
BrowserMatch "MS FrontPage" redirect-carefully
BrowserMatch "^WebDrive" redirect-carefully
BrowserMatch "^WebDAVFS/1.[0123]" redirect-carefully
BrowserMatch "^gnome-vfs/1.0" redirect-carefully
 </Directory>
```

Détaillons un peu plus ces lignes : la première ligne définit un verrou que va poser Apache, nécessaire pour que WebDAV fonctionne.

Nous voyons, de plus, que nous demandons à Apache de monter le répertoire **`/tmp`** sur l'URL « webdav ».

Observons plus précisément deux commandes : la première commande, **`Dav on`**, dit au serveur Apache que cette URL doit être servie avec le protocole WebDAV. La seconde, **`BrowserMath`** signifie que si le nom du client WebDAV correspond à **`X`** (remplacer **`X`** par un nom de client), il faut alors gérer la redirection de façon « sympathique ». Autrement dit, on signale à Apache les clients WebDAV ayant buggé !

Lors du redémarrage du serveur Apache et en faisant pointer un navigateur sur localhost/webdav, on doit apercevoir une arborescence de file system. L'autre test ultime est de monter cette URL avec un explorateur de bureau (prendre l'explorateur de GNOME).

5.2 WebDAV comme file system H2

Nous allons maintenant créer une base de données locale avec H2, soit quelques tables. Enfin, nous copierons la base de données dans le répertoire du WebDAV.

Le but désormais est de coder pour H2 le file system WebDAV. Pour ce faire, nous allons utiliser la librairie **`common-vfs`** **[5]** de la fondation Apache pour la partie protocolaire. En somme, H2 va appeler notre librairie qui va piloter la librairie **`common-vfs`**. Celle-ci va alors appeler le serveur WebDAV.

Implémenter un système de fichiers pour H2 consiste en trois étapes : définition d'une arborescence de fichiers, description de ce qu'est un fichier et enfin, enregistrement dans H2 du système de fichiers. Une arborescence de fichiers est une instance d'objet héritant de la classe **FilePath** qui permet grossièrement de donner le nom du fichier, dire s'il s'agit d'un répertoire ou pas et enfin, de lister les sous-éléments s'il s'agit d'un répertoire. L'opération pour un objet **FilePath** d'ouverture de fichier est la seule à être un peu complexe. Cette ouverture génère un fichier **FileBase** qui modélise le fichier proprement dit. Ce dernier doit pouvoir être lu de manière aléatoire. Il doit également être écrit de façon aléatoire si la base de données n'est pas en lecture seule.

Avec ces deux éléments, **FilePath** et **FileBase**, il faut enregistrer le file system dans le système de H2. Notre base de données sera alors accessible via l'URL suivante : **jdbc:h2:vfs:webdav://localhost/webdav/h2/test;ACCESS_MODE_DATA=r**.

Cette URL H2 se comprend ainsi :

- La base de données est accessible via un système de fichiers VFS. Sur ce système de fichiers, elle est donc accessible via **webdav://localhost/webdav/h2/test**.
- La base de données est accessible en lecture seule (**ACCES_MODE_DATA=r**).

5.2.1 Les fichiers

Le but ici est donc de simuler une arborescence de fichiers (modélisée par les objets de classe **FilePath**) avec VFS.

Commençons par créer une classe **VFSFilePath** (pour **FilePath** sur VFS) avec :

- Un constructeur par défaut. Ce constructeur va servir à créer un objet pour l'enregistrer dans le serveur H2.
- Un constructeur prenant en compte un fichier de la librairie **common-vfs FileObject**.

Le but du constructeur par défaut est de pouvoir construire un premier objet **VFSFilePath** (une sorte de répertoire racine) : nous allons informer H2 de l'existence de cette nouvelle racine.

Avec le constructeur par défaut, nous allons pouvoir créer une nouvelle fonction **main** qui va servir à enregistrer notre file system et lancer H2 :

```
public static void main(String [] args) throws SQLException
  {
    //Enregistrement du système de fichiers VFS
    FilePath.register(new VFSFilePath());
    //lancement de H2
    org.h2.tools.Console.main(args);
  }
```

La méthode principale d'un file system H2 est dans la fonction **getPath**. Cette méthode doit prendre en compte un nom de fichier H2 et renvoyer un objet **FilePath**.

La seule difficulté ici est de comprendre que pour H2, tous nos fichiers sont de la forme **vfs :<protocole distant> :fichier**. Par exemple, dans le cas de WebDAV, **vfs :webdav:/test.db**.

Or, pour la librairie **common-vfs**, les fichiers sont de la forme **<protocole distant>:/fichier** (et donc, pour l'exemple : **webdav:/test_db**). Notre méthode consiste donc à devoir prendre un nom de fichier H2, supprimer le mot **vfs** et demander à **common-vfs** de résoudre le fichier. C'est ce que nous faisons dans ces quelques lignes :

```
@Override
  public VFSFilePath getPath(String path) {
    //remove scheme
    String withoutScheme=path.substring(4, path.length());
    try{FileSystemManager fsManager = VFS.getManager();
    return new VFSFilePath(fsManager.resolveFile(withoutScheme)) ;}
catch (FileSystemException e) {
     throw DbException.convert(e);
    }
  }
```

Pour les mêmes raisons (différence de noms de fichiers entre H2 et **common-vfs**), notre constructeur **VFSFilePath(FileObject)** ressemble à :

```
private FileObject obj;

  VFSFilePath(FileObject obj)
  {
    this.obj=obj;
    super.name="vfs:"+obj.toString();
  }
```

Enfin, il y a encore trois méthodes importantes à mettre en place, les autres étant assez triviales à écrire :

- **List<FilePath> newDirectoryStream()** qui renvoie dans le cas d'un répertoire les éléments « enfants » ;
- **String getScheme()** qui renvoie le « nom » que prend en charge notre moteur ;
- **FileChannel open(String mode) throws IOException** qui renvoie un objet modélisant le contenu d'un fichier.

L'objectif de la méthode **newDirectoryStream** est pour un objet de type **FilePath** (qui représente un répertoire) de renvoyer les fichiers « enfants » de ce répertoire. Afin d'atteindre ce but, nous allons demander à l'objet **FileObject** de VFS de lister ses potentiels fichiers « enfants ». Puis, avec chaque enfant, nous allons l'encapsuler dans un objet **VFSFilePath** :

```
FileObject [] childs=this.obj.getChildren();
List<FilePath> result=new ArrayList<FilePath>();
if (childs!=null) { for (FileObject child:childs) result.add(new
VFSFilePath(child));}
return result;
```

La deuxième méthode est fondamentale et très simple à implémenter :

```
String getScheme(){return "vfs";}
```

Malgré sa simplicité, cette méthode est essentielle. En effet, lorsque H2 va voir l'URL de la base de données (ici **vfs :webdav://...**), le moteur va sélectionner le préfixe de l'URL (ici **vfs**) et challenger tous les systèmes de fichiers connus, afin de savoir qui peut le (ou la) prendre en charge. Autrement dit, lorsque H2 voit le préfixe **vfs**, il demande aux moteurs enregistrés lequel a une méthode **getScheme** renvoyant **vfs**. Lorsque celui-ci le trouve, il invoque avec cette URL la méthode **getPath** présentée plus haut.

Enfin, la dernière méthode, intitulée **open**, a pour but de prendre un mode d'ouverture (dans notre cas, un mode en lecture seule uniquement) et de renvoyer le contenu des fichiers :

```
public FileChannel open(String mode) throws
IOException
{return new VFSFileBase(obj);}
```

5.2.2 Le contenu des fichiers

Le contenu des fichiers est modélisé dans notre exemple par un objet **VFSFileBase** étendant l'objet H2 **FileBase**. Les instances de cette classe doivent pour H2 modéliser le contenu d'un fichier. Comme longuement expliqué plus haut, il est de bon ton que ce fichier soit accessible a minima en lecture aléatoire et au mieux en lecture/écriture aléatoire.

Pour notre système WebDAV, le process va être relativement simple, VFS fournissant un objet **RandomAccessContent** qui modélise déjà le contenu d'un fichier en lecture et/ou en écriture aléatoire.

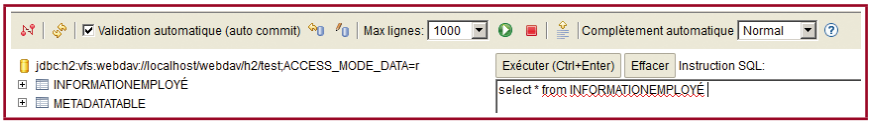
Notre principal travail consiste donc à récupérer un objet **RandomAccessContent** avec **nom de fichier FileObject**. C'est ce que fait le petit morceau de code suivant :

```
private RandomAccessContent randomAccess;
  public VFSFileBase(FileObject obj) {
    try {
    this.content=obj;
    this.randomAccess=obj.getContent().getRa
ndomAccessContent(RandomAccessMode.READ);
} catch (FileSystemException e) {}}
```

5.2.3 Test du système WebDAV

Afin de tester notre système avec WebDAV, il faut d'abord créer une base de données en local. Il suffit ensuite de copier ces fichiers sur le serveur WebDAV et de rouvrir la base de données, soit en lecture seule, soit en utilisant la bonne URL, à savoir, pour une base de données test hébergée sur un serveur localhost : **jdbc:h2:vfs:webdav://localhost/webdav/h2/test;ACCESS_MODE_DATA=r**.

Le moteur H2 doit alors effectuer les connexions et afficher la totalité des tables (en lecture seule), comme si elles étaient en local :



Affichage des tables en lecture seule et en mode WebDAV

6 Row/Cell Level Security

La mise en place de notre « Row/Cell Level Security » va passer par trois acteurs :

- Le premier acteur est une table décrivant les tables et les colonnes qui sont « à protéger ». Cette table s'appellera **MetaDataTable** et existera pour chaque base de données contenant des tables sécurisées ;
- Le deuxième acteur est un trigger qui va être déclenché lorsque l'administrateur va inclure une ligne dans la table **MetaDataTable**. En effet, nous ne pouvons pas sécuriser n'importe quelle colonne de n'importe quelle table. Nous mettons comme condition que, pour qu'une colonne puisse être sécurisée, il faut que celle-ci soit une colonne de type **VARCHAR** (c'est une simplification dans le cadre de l'article), mais surtout, qu'elle ait la propriété nullable. Il peut en effet être stipulé dans la clause de sécurité qu'en fonction du login d'un utilisateur, une colonne ne contienne que des valeurs **NULL** ;
- Le troisième acteur est un moteur de tables ressemblant exactement à un moteur de tables standard, mais qui invoquera avant de renvoyer des résultats notre table **MetaDataTable**.

Voyons un peu dans l'exemple ci-dessous comment vont se comporter nos acteurs. Nous avons une table **InformationEmployé** contenant un employé très important, Nathanaël :

ID	NAME	SURNAME	TELEPHONE	BUREAU
0	GP	Nathanaël	'0033685569090'	B12

Notre base de données contient deux utilisateurs RH (pour ressources humaines) et SA (login générique de l'administrateur). Parmi ces données, le numéro de téléphone doit être protégé : seul l'utilisateur RH peut le lire. Bien évidemment, la colonne **Téléphone** possède la propriété nullable et c'est le rôle de notre trigger de s'en assurer.

C'est ce que l'on retrouve dans la table **MetaDataTable** :

ID	TABLENAME	COLUMN-NAME	JAVASCRIPT
1978413685	INFORMATIONEM-PLOYÉ	TELEPHONE	**var resultValue=(user=="RH")? originalValue:"XX"**

Ces données se comprennent ainsi : pour la table **InformationEmployé** et en particulier pour la colonne **Téléphone**, la règle de sécurité **var resultValue=(user=="RH")?originalValue:"XX"** doit être appliquée. Dans le cas où

l'utilisateur n'est pas RH, le moteur va ainsi remplacer la valeur d'origine du numéro de téléphone par la valeur **XX**.

6.1 Mise en place de table MetaDataTable

Le premier acteur à mettre en place est la table **MetaDataTable**. Cette table est un objet table H2 classique composé :

- d'un identifiant unique représentant la clef primaire,
- du nom de la table potentiellement sécurisé,
- de la colonne de la table potentiellement sécurisée,
- d'une colonne nommée « JavaScript » contenant un script de sécurisation.

En SQL, cet objet **MetaDataTable** se définit ainsi :

```
CREATE TABLE METADATATABLE(
ID INT,
TABLENAME VARCHAR,
COLUMNNAME VARCHAR,
JAVASCRIPT VARCHAR
)
```

La colonne JavaScript est la colonne la plus intéressante, parce que c'est elle qui contient l'intelligence de sécurisation. En pratique, elle contient un script dans le langage JavaScript qui va être invoqué pour la sécurisation. Souvenez-vous en effet que lorsque H2 va rencontrer une colonne sécurisée d'une table sécurisée, il va alors invoquer sur la valeur de la ligne le script JavaScript contenu dans la table **MetaDataTable**.

Le langage Java autorise à invoquer des scripts via la JSR 223 **[7]**. Plus précisément, le langage JavaScript est inclus « par défaut » dans le langage Java.

Nous allons donc créer une méthode qui va invoquer le script JavaScript contenu dans la table **MetaDataTable**. Pour ce faire, il faut :

- demander au langage Java de créer le moteur JavaScript,
- se connecter à la base H2 pour récupérer le code JavaScript,
- invoquer le code JavaScript sur des valeurs de colonnes sécurisées.

6.1.1 Créer un moteur JavaScript

Comme évoqué plus haut, le langage JavaScript est inclus dans le langage Java. Pour le créer, il faut donc dire au langage Java que nous avons besoin d'invoquer un moteur de scripts via un manager de langage de script :

```
ScriptEngineManager mgr = new ScriptEngineManager();
```

Puis, avec ce manager de langage de script, nous demandons un moteur JavaScript que nous sauvegardons dans un attribut **engine** :

```
engine = mgr.getEngineByName("javascript");
```

6.1.2 Connexion à la base H2 et récupération du code JavaScript

Nous voulons maintenant pouvoir récupérer le code JavaScript contenu dans la table **MetaDataTable**. Il y a une chose amusante : nous avons ici le besoin de récupérer des données dans le moteur H2. En effet, le code que nous écrivons se trouve « dans » le moteur H2, et nous avons besoin de récupérer des données dans H2. La solution est d'établir une connexion JDBC entre notre code et le moteur H2, ce qui peut se faire via le code suivant :

```
public Connection getConnection(Session session)
{return session.createConnection(false);}
```

Cette méthode prend en compte une session H2 qui représente la connexion entre l'utilisateur de H2 et le moteur de H2. Cet objet est présenté par H2 à tous les moteurs de tables lors de l'exécution d'une requête. La « session » permet d'obtenir pour un moteur de tables une requête JDBC vers le moteur H2 que nous utilisons, afin de récupérer le code JavaScript via une méthode **isRulesFor**. Celle-ci renvoie, pour une table et une colonne, un code JavaScript de sécurité (ou **NULL** si aucun code n'est présent).

De façon assez classique, nous utilisons un **SELECT** sur la colonne JavaScript de la table **MetaDataTable** avec les noms de la table et de la colonne potentiellement sécurisés :

```
public String isRulesFor(Session session,String tableName,String
columnName)
  {
    final String sql="select javascript from METADATATABLE where
TABLENAME=? AND COLUMNNAME=?";
    String result=null;
    try{Connection c=getConnection(session);
    PreparedStatement statement = c.prepareStatement(sql);
    statement.setString(1,tableName);
    statement.setString(2,columnName);
    ResultSet resultSet = statement.executeQuery();
    while(resultSet.next()) result=resultSet.getString(1);
    statement.close();
    }catch(SQLException e){}
    return result;
  }
```

6.1.3 Invocation du code JavaScript

Nous attaquons maintenant l'invocation du code JavaScript sur la valeur possible d'une ligne d'une table potentiellement sécurisée. Cette méthode se définit ainsi :

```
public Value executeScriptForValue(String user,Value v,int
waitedValueType,String script)
```

Elle prend en compte :

- le nom de l'utilisateur qui exécute une requête dans le moteur H2,
- la valeur **v** initiale de la ligne (si l'on reprend l'exemple précédent, le numéro de téléphone de Nathanaël),

- le type SQL de la valeur attendue de retour. Dans le contexte de cet article, le seul type possible est **VARCHAR**,
- le script de sécurisation.

Cette méthode doit retourner la valeur qui sera renvoyée à l'utilisateur par le moteur H2. Ceci est le résultat du filtrage de la valeur initiale **v** par le script de sécurité.

Pour ce faire, nous devons définir un « univers » de valeurs pour le script de sécurité, composé du nom de l'utilisateur, de la valeur initiale, du type de la valeur initiale, ainsi que du type attendu de retour :

```
Bindings bindings = engine.createBindings();
bindings.put("user", user); // nom de l'utilisateur
bindings.put("typeValue", v.getType()); //type initial de la valeur
bindings.put("originalValue", v.toString()); // la valeur sous forme de chaine de caractères
bindings.put("waitedType", waitedValueType); // le type attendu de retour
```

Le script de sécurité est exécuté avec le moteur JavaScript et son « univers » :

```
engine.eval(script, bindings);
```

Enfin, la valeur de retour est récupérée de l'« univers » via la variable **resultValue**. Si cette variable est **NULL**, nous renvoyons la valeur SQL **NULL** ou alors, nous renvoyons un **VARCHAR** contenant cette valeur :

```
Object result=bindings.get("resultValue");
if (result==null)  {return  ValueNull.INSTANCE;}
else {return ValueString.get(result.toString());}
```

6.2 Triggers

La table **MetaDataTable** étant mise en place, il va falloir surveiller les entrées de cette table. En effet, dans le contexte de cet article, nous avons décidé que seules les colonnes de type **VARCHAR** et pouvant contenir des valeurs **NULL** pouvaient être insérées.

Afin de vérifier cette hypothèse, il est possible de mettre en place un trigger sur la table **MetaDataTable**. Ce trigger fera ainsi une partie du travail, à savoir de vérifier que pour une table existante, une colonne sécurisée correspond bien à ce qui est attendu.

L'autre partie du travail sera d'empêcher l'altération de nos tables sécurisées. En effet, il ne faudrait pas qu'un petit malin s'amuse dans notre table **MedatDataTable** à rendre **NOT NULL** une colonne contenant des valeurs **NULL** via une instruction **ALTER TABLE**. Une solution assez drastique à ce problème serait de supprimer de nos tables la possibilité d'être altérées.

Un trigger dans H2 se définit comme un objet Java implémentant l'interface trigger. Dans cette interface, nous pouvons trouver une méthode :

```
  public void fire(Connection conn,Object[]
oldRow, Object[] newRow)
```

Cette méthode prend en compte une connexion JDBC vers le moteur H2, une potentielle « ancienne ligne » (dans le cas d'un trigger intervenant après une modification de table) et la nouvelle ligne de la table modifiée.

Dans le cas qui nous intéresse, notre trigger **MetaDataTriggers** doit être déclenché avant l'insertion dans la table **MetaDataTable**. Nous n'avons donc à manipuler que le paramètre **newRow**.

Le code de ce trigger est un code JDBC somme toute assez classique, qui commence par récupérer de la nouvelle ligne le nom de la table que nous insérons, puis le nom de la colonne. Avec ce couple, nous demandons à H2 d'obtenir les métadonnées du nom de la table en train d'être inséré :

```
String tableName=(String) newRow[1];
String columnName=(String) newRow[2];
ResultSet table = conn.getMetaData().
getColumns(null, null, tableName, columnName);
```

Enfin, nous validons :

- l'existence de la table :

```
if ( !table.next()) throw new SQLException(" la table n'existe pas ")
```

- le fait que la colonne soit de type **VARCHAR** :

```
if (table.getInt("DATA_TYPE")!=java.sql.
Types.VARCHAR)
{throw new SQLException("La colonne
("+columnName+") n'est pas une chaine de
caractères");}
```

- le fait que la colonne ait la propriété **NULL** :

```
if (!table.getString("IS_NULLABLE").equals("YES"))
{throw new SQLException("La colonne ("+columnName+")
doit pouvoir être nullable");}
```

Enfin, ce trigger est ajouté sur la table **MetaDataTable** via la requête SQL suivante :

```
CREATE TRIGGER MYTRIGGER BEFORE INSERT, UPDATE ON
PUBLIC.METADATATABLE FOR EACH ROW CALL "com.
neuresys.tableengine.MetaDataTriggers"
```

6.3 Mise en place du moteur de tables sécurisées

Un moteur de tables dans H2 est un objet implémentant l'interface **TableEngine**. Cette dernière contient une méthode autorisant un moteur de tables à renvoyer un objet représentant une table via :

```
public TableBase createTable(CreateTableData data)
```

Cette méthode prend en compte les informations de création de tables (en gros, une structure JAVA encapsulant les informations d'une requête **CREATE TABLE**) et renvoie une instance d'objet **TableBase**.

La hiérarchie des objets Table dans H2 se décrit ainsi :

- Une table H2 est une instance d'objet abstrait **Table** ;
- Une **TableBase** est la descendante directe de **Table** et implémente les opérations basiques d'une table ;
- Une table « standard » créée via un **CREATE TABLE** se traduit en interne par l'instanciation d'un objet **RegularTable**, descendant direct de **TableBase**.

La question est de savoir comment créer notre table sécurisée, que l'on appelle aussi **SecureTable**. Il s'agit en fait quasiment d'une table H2 normale, sauf trois petites choses :

- Nous appliquons nos règles de sécurité sur les valeurs retournées ;

- Nous n'autorisons pas H2 à mettre en cache les valeurs retournées (pour prendre en compte un éventuel changement de script de sécurité) ;
- Nous n'autorisons pas les commandes **ALTER** sur nos tables.

A cause de la ressemblance entre notre **SecureTable** et les **RegularTable** de H2, nous décidons que nos objets vont hériter de **RegularTable**. C'est dans cet héritage que nous allons ajouter nos modifications.

La première modification est triviale et concerne le problème de cache. Pour que H2 puisse mettre des valeurs en cache, il faut ainsi que la table ait une propriété « déterministe ». Cette propriété indique que lorsque deux requêtes sont identiques sur une table non modifiée entre ces deux requêtes, celles-ci renvoient alors les mêmes résultats.

En signalant que notre table ne possède pas cette propriété, les résultats ne seront donc pas mis en cache.

Cela se traduit pour notre classe **SecureTable** par :

```
public boolean isDeterministic() {return false;}
```

La deuxième modification est tout aussi simple. Pour interdire les requêtes **ALTER**, il faut créer une méthode **checkSupportAlter** qui renvoie une exception de type **Runtime**.

Enfin, la troisième modification est elle aussi assez simple, mais elle nécessite de comprendre une partie de la hiérarchie de classes de H2.

Lorsque l'on fait un **SELECT** sur une table H2, ce scan de la table se traduit par la création d'un objet **Index**. Une table peut être ainsi considérée comme un ensemble de pages à lire, dont l'ordre de lecture peut changer : soit on le lit de la première à la dernière page et l'on parle alors d'un « scan table » classique, soit, si l'on est uniquement intéressé par les têtes de chapitre, on ne lit que les pages « chapitres ». L'objet **Index** représente cet ordre de lecture par chapitres. Lorsqu'on crée un objet **Index**, il faut donc créer un objet qui représente la page courante de lecture. Cet objet nommé **Cursor** peut ainsi nous dire si l'on est en train de lire la dernière page et si l'on peut changer de page.

Enfin, un **Cursor** permet d'avoir un objet **SearchRow** qui représente la valeur de la page courante (en somme, c'est avec cet objet que l'on lit réellement).

Notre premier travail est donc :

- De faire hériter nos objets **SecureTable** de **RegularTable**, un objet **SecureIndex** de **Index**, un objet **SecureCursor** de **Cursor** et enfin, un objet **SecureSearchRow** de **SearchRow** ;
- Par délégation, les méthodes surchargées renvoient nos objets **Secure**.

Afin de mieux comprendre ce pattern (qui passe assez mal à l'écrit), voici ci-dessous un petit exemple ; la classe **SecureIndex** implémente l'interface **Index** et effectue une délégation sur un objet **Index** :

```
public class SecureIndex  implements Index
{
  Index index;
  private SecureTable secureTable;
public SecureIndex(SecureTable secureTable, Index scanIndex) {
  this.index=scanIndex;
  this.secureTable=secureTable;
  }
public Schema getSchema() {
    return index.getSchema();
  }
```

Et la classe **SecureTable** fait la même chose :

```
public class SecureTable extends RegularTable{
public SecureTable(CreateTableData data) {super(data);}
  @Override
  public Index getScanIndex(Session session) {
    // TODO Auto-generated method stub
    return new SecureIndex(this,super.getScanIndex(session));
  }
```

Cet objet **Index** permet ensuite la création d'un objet de type **Cursor** qui représente une ligne d'une table. Enfin, notre classe **SecureIndex** renvoie avec le même pattern des instances de classes **SecureSearchRow** qui modélisent les valeurs d'une ligne :

```
public SearchRow getSearchRow()
{return new SecureSearchRow(secureTable,underObject.
getSearchRow(),this.session);}
```

Les implémentations de **SearchRow** permettent donc de lire les valeurs d'une ligne. Parmi les quelques méthodes de cette interface, une nous intéresse particulièrement : **getValue(int index)**. Cette dernière se doit de renvoyer au moteur H2 la valeur contenue dans l'**Index** d'une ligne de table.

C'est exactement à cet endroit que nous devons intervenir pour effectuer la liaison avec le JavaScript.

La méthodologie est simplissime : nous récupérons le nom de la colonne de l'**Index** de la table. Nous demandons s'il existe une règle de sécurité pour la table et cette colonne, et si oui, nous invoquons le code JavaScript :

```
public Value getValue(int index) {
Column c=secureTable.getColumn(index);
Value v=searchRow.getValue(index);
String rules=SecureUtils.INSTANCE.isRulesFor(session, secureTable.
getName(), c.getName());
if (rules!=null)
 {
 v=SecureUtils.INSTANCE.executeScriptForValue(session.getUser().
getName(), v, c.getType(), rules);
}
return v;
}
```

Après avoir recréé une hiérarchie de tables **Index**, **Cursor** et **SearchRow** via un pattern de délégations et une fois la liaison avec le JavaScript effectuée, il ne reste plus qu'à expliciter notre classe implémentant **TableEngine** :

```
package com.neuresys.tableengine
import org.h2.command.ddl.CreateTableData;
import org.h2.table.TableBase;
public class SecureTableEngine implements org.h2.api.TableEngine {
  @Override
  public TableBase createTable(CreateTableData data) {
    return new SecureTable(data);
  }
}
```

6.4 Test du système

Une fois tout installé, nous allons pouvoir faire un petit test du système après s'être assuré d'avoir au moins deux utilisateurs RH (pour ressources humaines) et SA (administrateur système : ce compte est créé par défaut).

La table d'information employée est définie comme une table « normale ». Nous précisons cependant à H2 que cette table doit utiliser notre **SecureTableEngine** :

```
CREATE TABLE INFORMATIONEMPLOYÉ(ID INT,NAME VARCHAR,SURNAME
VARCHAR,TELEPHONE VARCHAR,BUREAU VARCHAR)
ENGINE "com.neuresys.tableengine.SecureTableEngine"
```

Comme précisé plus haut, cette table comprend les valeurs suivantes :

ID	NAME	SURNAME	TELEPHONE	BUREAU
0	GP	Nathanaël	'0033685569090'	B12

La table **metadatable** comprend quant à elle : (voir tableau en bas de page).

Effectuons maintenant deux simples scans de la table employée avec l'utilisateur SA et l'utilisateur RH :

ID	NAME	SURNAME	TELEPHONE	BUREAU
0	GP	Nathanaël	'XX'	B12

*Exécution de la requête select * from INFORMATIONEMPLOYÉ avec l'utilisateur SA*

ID	NAME	SURNAME	TELEPHONE	BUREAU
0	GP	Nathanaël	'0033685569090'	B12

Exécution de la requête avec l'utilisateur RH

Conclusion

Dans cet article, nous avons exploré une infime partie du moteur H2 et effectué deux modifications.

La première modification favorise l'accès à des bases de données distantes (via du WebDAV). Au-delà de cet accès, nous avons voulu démontrer qu'il était possible de séparer physiquement le moteur de bases de données (H2 en l'occurrence) de la base de données en elle-même.

L'idée générale étant de pouvoir placer le moteur de bases de données, non pas de façon centralisée comme c'est généralement le cas, mais de façon décentralisée (un moteur par poste client).

L'autre modification apportée est l'ajout d'un module de Row/Cell Level Security à notre moteur H2. Ce module est juste une esquisse de ce qu'il est possible de faire fonctionnellement et nécessite beaucoup plus de travail pour être mis en production. On tâchera par exemple d'étendre ce module à la totalité des types SQL, de pousser les informations de sécurité vers l'optimiseur, voire l'analyseur (afin de prendre en compte par réécriture SQL les règles de sécurité renvoyant **NULL** pour certains utilisateurs), ou encore de mettre en place un cache pour éviter le re-calcul des règles de sécurité.

Mais la véritable conclusion est que mon petit Naël est rassuré désormais, il sait qu'avec de la bonne volonté, l'arrivée de « petit frère » se passera très bien ! ■

Remerciements

Cet article est d'abord une façon plus ou moins adroite de souhaiter la bienvenue à un petit bonhomme qui devrait naître fin juillet. Je voudrais remercier mon amour, Lisa : sans sa patience et sa compréhension, cet article n'aurait jamais pu voir le jour ! À l'image de mon petit Naël, l'arrivée prochaine de « petit frère » remue en effet pas mal votre humble serviteur, ce qui risque d'expliquer une certaine « absence » dans ces colonnes.

Références

[1] Implementing Row- and Cell-Level Security in Classified Databases, http://www.microsoft.com/technet/prodtechnol/sql/2005/multisec.mspx#E3MAC

[2] Protecting Your Data with Row Level Security for SQL Server Databases http://www.drdobbs.com/database/protecting-your-data-with-row-level-secu/215900773

[3] SQLite Virtual Table : http://www.sqlite.org/vtab.html

[4] SQLite file format : http://www.sqlite.org/fileformat2.html

[5] Common-vfs : http://commons.apache.org/proper/commons-vfs/

[6] http://code.google.com/p/gears/wiki/ContentRangePostProposal

[7] JSR223 :http://www.jcp.org/en/jsr/detail?id=223

ID	TABLENAME	COLUMNNAME	JAVASCRIPT
1978413685	INFORMATIONEMPLOYÉ	TELEPHONE	`var resultValue=(user=="RH")?originalValue:"XX"`

LE COIN DU VIEUX BARBU

par David Odin [Si barbu qu'il se méfie des tours en avion]

La dernière fois, nous avons décortiqué un tableur ayant gagné l'IOCCC [1] en 2000. Aujourd'hui, nous allons analyser le complément idéal : un simulateur de vol, gagnant de l'IOCCC [1] en 1998.

1 Ça plane pour moi !

{Je partais confiant, je vous assure. Tout semblait possible, comme d'habitude. Et pourtant...}

Mais commençons par le commencement. Aujourd'hui, nous allons décortiquer une nouvelle entrée de l'IOCCC **[1]**, vainqueur en 1998 dans la catégorie « Best of Show », qui implémente un simulateur de vol.

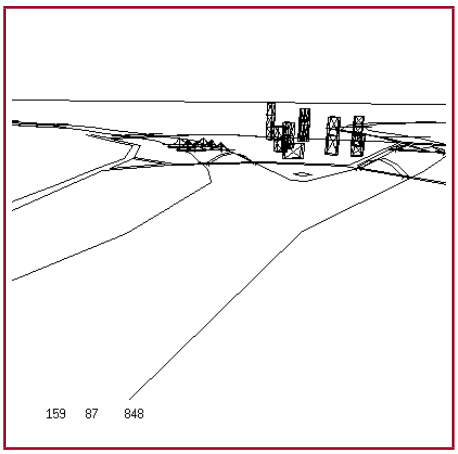


Fig. 1 : Un survol de Pittsburgh

Note

NDLR : Dans le présent article, vous remarquerez la présence de texte en italique entre accolades. Il s'agit là de l'expression du moi intérieur de l'auteur, qui s'est invité dans le processus d'écriture de l'article, afin de traduire les émotions et les souffrances de ce dernier. Merci de votre compréhension.

2 Version initiale

Comme vous pouvez le voir sur la figure 1, il s'agit d'un simulateur de vol à la première personne, avec une représentation du paysage (Pittsburgh ici) en mode filaire. Les indications de vitesse (en nœuds), de direction et d'altitude sont affichées en bas à gauche de la fenêtre. D'après l'auteur, c'est un Piper Cherokee qui est simulé. Il nous indique également que l'avion est considéré dans le code comme un solide à six degrés de liberté (on peut alors espérer trouver 3 coordonnées x, y, z et 3 angles dans le code).

Voici la version proposée à l'IOCCC **[1]**, sans changement :

```
#include                                  <math.h>
#include                                <sys/time.h>
#include                                <X11/Xlib.h>
#include                               <X11/keysym.h>
                                        double L ,o ,P
                                       ,_=dt,T,Z,D=1,d,
                                       s[999],E,h= 8,I,
                                       J,K,w[999],M,m,O
                                      ,n[999],j=33e-3,i=
                                      1E3,r,t, u,v ,W,S=
                                      74.5,l=221,X=7.26,
                                      a,B,A=32.2,c, F,H;
                                      int N,q, C, y,p,U;
                                     Window z; char f[52]
                                   ; GC k; main(){ Display*e=
 XOpenDisplay( 0); z=RootWindow(e,0); for (XSetForeground(e,k=XCreateGC (e,z,0,0),BlackPixel(e,0))
; scanf("%lf%lf%lf",y +n,w+y, y+s)+1; y ++); XSelectInput(e,z= XCreateSimpleWindow(e,z,0,0,400,400,
0,0,WhitePixel(e,0) ),KeyPressMask); for(XMapWindow(e,z); ; T=sin(O)){ struct timeval G={ 0,dt*1e6}
; K= cos(j); N=1e4; M+= H*_; Z=D*K; F+=_*P; r=E*K; W=cos( O); m=K*W; H=K*T; O+=D*_*F/ K+d/K*E*_; B=
sin(j); a=B*T*D-E*W; XClearWindow(e,z); t=T*E+ D*B*W; j+=d*_*D-_*F*E; P=W*E*B-T*D; for (o+=(I=D*W+E
*T*B,E*d/K *B+v+B/K*F*D)*_; p<y; ){ T=p[s]+i; E=c-p[w]; D=n[p]-L; K=D*m-B*T-H*E; if(p [n]+w[ p]+p[s
]== 0|K <fabs(W=T*r-I*E +D*P) |fabs(D=t *D+Z *T-a *E)> K)N=1e4; else{ q=W/K *4E2+2e2; C= 2E2+4e2/ K
 *D; N-1E4&& XDrawLine(e ,z,k,N ,U,q,C); N=q; U=C; } ++p; } L+=_* (X*t +P*M+m*l); T=X*X+ l*l+M *M;
  XDrawString(e,z,k ,20,380,f,17); D=v/l*15; i+=(B *l-M*r -X*Z)*_; for(; XPending(e); u *=CS!=N){
                                  XEvent z; XNextEvent(e ,&z);
                                      ++*((N=XLookupKeysym
                                        (&z.xkey,0))-IT?
                                        N-LT? UP-N?& E:&
                                        J:& u: &h); --*(
                                        DN -N? N-DT ?N==
                                        RT?&u: & W:&h:&J
                                         ); } m=15*F/l;
                                         c+=(I=M/ l,l*H
                                         +I*M+a*X)*_; H
                                         =A*r+v*X-F*l+(
                                         E=.1+X*4.9/l,t
                                         =T*m/32-I*T/24
                                          )/S; K=F*M+(
                                          h* 1e4/l-(T+
                                          E*5*T*E)/3e2
                                          )/S-X*d-B*A;
                                          a=2.63 /l*d;
                                         X+=( d*l-T/S
```

```
                                *(.19*E +a
                                *.64+J/1e3
                                )-M* v +A*
                                Z)*_; l +=
                                K *_; W=d;
                                sprintf(f,
                                "%5d  %3d"
                                "%7d",p =l
                               /1.7,(C=9E3+
                         0*57.3)%0550,(int)i); d+=T*(.45-14/l*
                        X-a*130-J* .14)*_/125e2+F*_*v; P=(T*(47
                        *I-m* 52+E*94 *D-t*.38+u*.21*E) /1e2+W*
                        179*v)/2312; select(p=0,0,0,0,&G); v-=(
                         W*F-T*(.63*m-I*.086+m*E*19-D*25-.11*u
                          )/107e2)*_; D=cos(o); E=sin(o); } }
```

Vous pouvez retrouver ce source sur la page http://forma3dev.fr/vieux-barbu/ (**banks.c**). Le moins que l'on puisse dire, c'est que le code est auto-documenté, au moins dans la forme ! L'auteur (Carl Banks) nous indique que pour compiler son programme, il convient de passer les noms des touches utilisées dans des prédéfinitions :

- **IT** doit contenir le nom de la touche permettant d'accélérer (**XK_Page_Up** par exemple),
- **DT** doit contenir le nom de la touche permettant de décélérer,
- **FD** la touche permettant de pousser le manche vers l'avant,
- **BK** pour tirer le manche en arrière,
- **LT** pour le pencher vers la gauche,
- **RT** pour le pencher vers la droite,
- **CS** pour le recentrer,
- **dt** doit contenir l'intervalle de temps entre deux affichages (et deux mises à jour des paramètres de vol).

Ainsi, pour compiler ce programme, l'auteur nous conseille la ligne suivante : **gcc banks.c -DIT=XK_Page_Up -DDT=XK_Page_Down -DUP=XK_Up -DDN=XK_Down -DLT=XK_Left -DRT=XK_Right -DCS=XK_Return -Ddt=0.02 -lm -lX11 -o banks**

Pour la suite de cet article, je placerai les définitions directement en début de programme, ce qui rendra la compilation plus simple, ainsi que les modifications.

L'auteur *{qui cherche manifestement à nous amadouer}* nous indique même comment utiliser son programme avec des jeux de données. Par exemple, la capture d'écran de la figure 1 a été faite en lançant le programme ainsi : **cat horizon.sc pittsburgh.sc | ./banks**.

Là encore, l'auteur *{qui commence à un peu trop montrer qu'il est sûr de lui}* nous indique que son programme lit sur l'entrée standard les coordonnées des points 3D qui seront reliés par des lignes. Une rupture de ligne est indiquée par le point **(0, 0, 0)**. Cela devrait nous aider à comprendre le programme. Des fichiers **.sc** sont disponibles sur mon site, tout comme sur celui de l'IOCCC **[1]**.

{À ce point de mon analyse, je trouvais encore l'auteur assez sympathique, j'étais encore prêt à croire que la mise en forme d'avion du code devait être la principale source d'obfuscation et que cela avait donné plein de contraintes. Je croyais encore que l'auteur avait subi tout cela et avait dû s'adapter. J'ignorais à cet instant qu'au contraire, il s'en délectait !}

3 Remise en forme

Nos fidèles lecteurs qui commencent à me connaître savent qu'une de mes armes favorites est la remise en forme du code. Pour cela, je commence par tout mettre sur une seule ligne, puis je parcours cette ligne en mettant des espaces avant les parenthèses ouvrantes, après les virgules et autour des opérateurs, et des retours à la ligne après les points-virgules et avant et après les accolades. Il y a quelques exceptions à cette démarche, mais elles sont rares. On arrive facilement à la version suivante : (**banks.reformat.c**)

```
#include <math.h>
#include <sys/time.h>
#include <X11/Xlib.h>
#include <X11/keysym.h>

#define IT XK_Page_Up
#define DT XK_Page_Down
#define UP XK_Up
#define DN XK_Down
#define LT XK_Left
#define RT XK_Right
#define CS XK_Return
#define dt 0.02

double L ,o ,P, _ = dt, T, Z, D = 1, d, s[999], E,
h = 8, I, J, K;
double w[999], M, m, O, n[999], j = 33e-3, i = 1E3,
r, t, u, v, W;
double S = 74.5, l = 221, X = 7.26, a, B, A = 32.2,
c, F, H;
int N, q, C, y, p, U;
Window z;
char f[52];
GC k;

main()
{
  Display *e = XOpenDisplay(0);
  z = RootWindow(e, 0);
  for (XSetForeground(e, k = XCreateGC(e, z, 0, 0),
BlackPixel(e, 0));
       scanf("%lf%lf%lf", y + n, w + y, y + s) + 1;
       y++)
       ;
  XSelectInput(e, z = XCreateSimpleWindow(e, z, 0,
0, 400, 400, 0, 0,

WhitePixel(e, 0)),
               KeyPressMask);
  for (XMapWindow(e, z); ; T = sin(O))
  {
    struct timeval G = { 0, dt * 1e6 };
    K = cos(j);
    N = 1e4;
    M += H * _;
    Z = D * K;
    F += _ * P;
    r = E * K;
```

```
    W = cos(O);
    m = K * W;
    H = K * T;
    O += D * _ * F / K + d / K * E * _;
    B = sin(j);
    a = B * T * D - E * W;
    XClearWindow(e, z);
    t = T * E + D * B * W;
    j += d * _ * D - _ * F * E;
    P = W * E * B - T * D;
    for (o += (I = D * W + E * T * B,
               E * d / K * B + v + B / K * F * D)
* _; p < y; )
    {
      T = p[s] + i;
      E = c - p[w];
      D = n[p] - L;
      K = D * m - B * T - H * E;
      if (p[n] + w[p] + p[s] == Ø |
          K < fabs(W = T * r - I * E + D * P) |
          fabs(D = t * D + Z * T - a * E) > K)
        N = 1e4;
      else
      {
        q = W / K * 4E2 + 2e2;
        C = 2E2 + 4e2 / K * D;
        N - 1E4 && XDrawLine(e ,z, k, N ,U, q,C);
        N = q;
        U = C;
      }
      ++p;
    }
    L += _ * (X * t + P * M + m * l);
    T = X * X + l * l + M * M;
    XDrawString(e, z, k, 2Ø, 38Ø, f, 17);
    D = v / l * 15;
    i += (B * l - M * r - X * Z) * _;
    for (; XPending(e); u *= CS != N)
    {
      XEvent z;
      XNextEvent(e, &z);
      ++*((N = XLookupKeysym(&z.xkey, Ø)) - IT ?
      N - LT ? UP - N ? &E : &J : &u : &h);
      --*(DN - N ? N - DT ? N == RT ? &u: &W : &h
: &J);
    }
    m = 15 * F / l;
    c += (I = M / l, l * H + I * M + a * X) * _;
    H = A * r + v * X - F * l + (E = .1 + X * 4.9 / l,
                                 t = T * m / 32 - I *
T / 24) / S;
    K = F * M + (h * 1e4 / l - (T + E * 5 * T * E)
/ 3e2)
        / S - X * d - B * A;
    a = 2.63 / l * d;
    X += (d * l - T / S * (.19 * E + a * .64 + J /
1e3) -
          M * v + A * Z) * _;
    l += K * _;
    W = d;
    sprintf(f, "%5d  %3d" "%7d",p = l /1.7,
            (C = 9E3 + O * 57.3) % 0550, (int)i);
    d += T *(.45 - 14 / l * X - a * 13Ø - J * .14) *
_ / 125e2 +
             F * _ * v;
    P = (T *(47 * I - m * 52 + E * 94 * D - t * .38 +
u * .21 * E)
         / 1e2 + W * 179 * v) / 2312;
    select(p = Ø, Ø , Ø , Ø, &G);
    v -= (W * F - T * (.63 * m - I * .Ø86 + m * E
* 19 -
          D * 25 - .11 * u) / 107e2) * _;
    D = cos(o);
      E = sin(o);
    }
}
```

Nous voyons apparaître la structure générale du programme : une seule fonction, quelques initialisations et une boucle sans fin.

L'utilisation de variables globales ne devrait pas être un problème pour la compréhension, surtout qu'elles sont en nombre réduit. *{Nous verrons bientôt qu'il s'agit en fait d'une partie de l'horrible machination de Carl Banks !}*

4 La lecture des fichiers de données

On sait que le programme commence par lire les données qu'on lui présente sur l'entrée standard.

En cherchant un peu dans le code, on voit que cela ne peut être fait que par l'appel à la fonction **`scanf()`** dans la séquence suivante :

```
  for (XSetForeground(e, k = XCreateGC(e, z, Ø, Ø),
BlackPixel(e, Ø));
       scanf("%lf%lf%lf", y + n, w + y, y + s) + 1;
       y++)
       ;
```

Les appels de fonction dans la partie initialisation de la boucle **`for`** ne nous concernent pas pour l'instant, ils ne sont de toute façon exécutés qu'une fois avant le premier test de la boucle. Il reste un appel de **`scanf()`** dans le test de la boucle, et une incrémentation de la variable **`y`**. On peut alors transformer cette boucle ainsi :

```
while (scanf("%lf%lf%lf", y + n, w + y, y + s) + 1)
  y++;
```

Cette boucle tourne tant que **`scanf(...) + 1`** est vrai, donc non nul, et donc plus exactement tant que **`scanf()`** ne renvoie pas -1, ce qui est la valeur de **`EOF`** (*end of file*). L'appel à **`scanf()`** lit trois **`double`** comme spécifié par la chaîne de format **`"%lf"`**. Ces doubles sont rangés respectivement aux adresses **`y+n`**, **`w+y`** et **`y+s`**. Or **`n`**, **`w`** et **`s`** sont des tableaux de **`double`**. Et **`y`** est un compteur entier : il compte le nombre de triplets lus par le **`scanf()`** et peut donc être renommé **`nb_vertices`**. **`n`** est donc un tableau qui contiendra les coordonnées **`x`** des lignes dessinées, on peut donc renommer **`n`** en **`vertices_x`**, et **`y+n`** peut être réécrit **`&vertices_x[nb_vertices]`**. En renommant de la sorte les tableaux **`w`** et **`s`**, on a alors le code suivant, qui manifestement permet la lecture des données depuis les fichiers passés comme **`pittsburgh.sc`** :

```
  while (scanf("%lf%lf%lf",
         &vertices_x[nb_vertices],
         &vertices_y[nb_vertices],
         &vertices_z[nb_vertices]) != EOF)
    nb_vertices++;
```

5 Un peu de Xlib

Si l'on place la partie lisant les données en début de la fonction **`main()`**, on a ensuite le code suivant :

```
  Display *e = XOpenDisplay(Ø);
  z = RootWindow(e, Ø);
  XSetForeground(e, k = XCreateGC(e, z, Ø,
Ø), BlackPixel(e, Ø));
  XSelectInput(e, z = XCreateSimpleWindow(e,
z, Ø, Ø, 4ØØ, 4ØØ, Ø, Ø,

WhitePixel(e, Ø)),

KeyPressMask);
  for (XMapWindow(e, z); ; T = sin(O))
  {
    ...
```

On remarque alors une séquence de code typique du début d'un programme utilisant directement la Xlib pour créer une fenêtre. Nous avons déjà vu ce genre de code lors du précédent article. Je vous renvoie au code de cet article et notamment au petit programme **`xlib-mini.c`** que vous trouverez sur la page http://forma3dev.fr/vieux-barbu/.

En renommant quelques variables de manière adéquate et en sortant les affectations des paramètres de fonctions, on arrive à ce morceau de code :

```
Display *display;

display = XOpenDisplay(NULL); // Connexion au serveur
root_window = RootWindow(display, 0);
// Création d'un contexte graphique affichant en noir.
gc = XCreateGC(display, root_window, 0, NULL);
XSetForeground(display, gc, BlackPixel(display, 0));
// Création  d'une fenêtre avec un fond blanc.
window = XCreateSimpleWindow(display, root_window, 0, 0, 400, 400,
                             0, 0, WhitePixel(display, 0));
// Cette fenêtre réagira aux événements clavier
XSelectInput(e, window, KeyPressMask);
// Affichage de la fenêtre
XMapWindow(display, window);
for ( ; ; T = sin(0))
{
  ...
```

Il s'agit donc juste de la création et de la paramétrisation de la fenêtre du programme.

Là encore, rien de bien méchant, à part les affectations dans les paramètres ou les appels de fonctions dans les initialisations du **for**. *{Mais cela n'est qu'une machination destinée à me mettre en confiance pour mieux me détruire.}*

6 La gestion des touches

Dans un programme graphique, après avoir créé la fenêtre, on entre dans une boucle infinie pour calculer des trucs, dessiner la fenêtre ou réagir à des événements. Ici, le seul événement possible est l'appui sur une touche du clavier. La partie du code qui gère les touches est beaucoup plus facile à trouver qu'à comprendre ! La voici :

```
for (; XPending(e); u *= CS != N)
{
  XEvent z;
  XNextEvent(display, &z);
  ++*((N = XLookupKeysym(&z.xkey, 0)) - IT ? N - LT ? UP - N ? &E :
&J : &u : &h);
  --*(DN - N ? N - DT ? N == RT ? &u: &W : &h : &J);
}
```

Le **for** est évidemment un **while** qui exécutera le contenu de la boucle (y compris le **u *= CS != N** à la fin) tant qu'il y aura des événements à traiter. *{On voit arriver la perversité de l'auteur qui n'hésite pas à réutiliser le nom de variable z, alors que ce nom était déjà pris pour la fenêtre}* Mais bon, ici, il s'agit d'une variable locale à la boucle, on va dire que ce n'est pas trop grave, on la renommera tout de même **event**.

Les lignes restantes sont nettement plus tordues. Déjà, elles utilisent la variable **N** qui va contenir le code de la touche pressée, alors qu'elle a manifestement un autre rôle ailleurs. Là encore, il faut vérifier que c'est sans conséquence et c'est le cas : **N** sera réaffecté à la valeur **1e4** lors de sa prochaine réutilisation. Par ailleurs, les variables **J**, **u** et **h** sont utilisées par adresse ici, alors qu'elles ne sont pas du tout modifiées dans tout le reste du code (mais elles sont lues plusieurs fois chacune). Les variables **E** et **W** sont elles aussi utilisées par adresse dans ce bout de code, et sont beaucoup utilisées par ailleurs, cependant elles ne sont pas réutilisées avant d'être réaffectées. Les changements qu'on leur fera ici seront effacés. *{Ce psychopathe réutilise des noms de variables dès qu'il le peut pour éviter qu'on puisse comprendre ce qu'il fait ! }*

Maintenant que l'on sait où on va au niveau des variables, on peut regarder les deux lignes qui ont l'air si étranges. Bien évidemment, le **N = XLookupKeysym(&z.xkey, 0)** peut être déplacé avant. Cette partie utilise l'événement **z** en un code de touche Xlib, tel que **XK_Return** par exemple. La ligne du **++** devient :

```
++*(N - IT ? N - LT ? UP - N ? &E : &J : &u : &h);
```

Soit, avec des parenthèses :

```
++*(N - IT ? (N - LT ? (UP - N ? &E : &J) : &u) : &h);
```

N - IT est vrai (non nul) si **N** est différent de **IT**, soit **XK_Page_Up**. Donc, si on a bien appuyé sur la touche **XK_Page_Up**, **N - IT** est nul (faux) et cette ligne devient : **++*(&h);** soit **++h;**.

Sinon, **N** est comparé à **LT** (**XK_Left**), dans ce cas l'expression devient **++u;**. Sinon, on compare enfin **N** à **UP** (**XK_Up**). En cas d'égalité, on a **++J;**, sinon **++E;**. L'incrémentation de **E** ne sert strictement à rien, mais l'opérateur **?:** nécessite d'avoir des clauses homogènes. L'auteur a donc décidé d'incrémenter une variable un peu au hasard *{Vous voyez bien qu'il n'est pas net ! }* dans le cas où aucune des trois touches citées n'a été pressée.

De la même façon, avec la ligne suivante, **J** est décrémenté si **N** vaut **DN** (**XK_Down**, **h** également si on a pressé **DT** (**XK_Page_Down**), et **u** si on a pressé **RT** (**XK_Right**). Vous aurez peut-être remarqué que cette fois-ci, la variable qui est décrémentée par défaut est **W**, et que l'auteur a utilisé des variantes, en inversant l'ordre des variables ou en utilisant l'opérateur **==**.

Une dernière comparaison est **u *= CS != N**. **CS != N** vaut 1 si on n'a pas appuyé sur **CS** (**XK_Return**). Dans ce cas, **u** est multiplié par 1 et reste donc inchangé. Sinon, **u** est multiplié par 0, ce qui recentre le manche à balai.

On peut alors réécrire le code de gestion des touches ainsi :

```
while (XPending(display))
{
  XEvent event;
  XNextEvent(display, &event);
  switch (XLookupKeysym(&event.xkey, 0))
  {
    case XK_Up: J++; break;
    case XK_Down: J--; break;
    case XK_Left: u++; break;
    case XK_Right: u--; break;
    case XK_Page_Up: h++; break;
    case XK_Page_Down: h--; break;
    case XK_Return: u = 0; break;
  }
}
```

Ce qui est tout de même plus lisible et ne modifie pas sauvagement des variables qui ne nous ont rien demandé ! On pourrait renommer les variables **J**, **u** et **h** maintenant qu'on sait que **J** représente la position avant/arrière du manche, **u** la position gauche/droite et **h** l'accélération, mais je n'ai pas trouvé de noms adéquats *{probablement l'un des premiers effets de la déchéance dans laquelle Banks veut me plonger !}*.

7 On dessine le tout

Une autre partie importante dans un programme graphique est bien évidemment l'affichage dans la fenêtre. En parcourant le programme, on ne trouve que 3 appels graphiques :

- **XClearWindow(display, window);** qui efface complètement la fenêtre,
- **XDrawString(...)** qui sert à afficher les informations de vitesse, direction et d'altitude ; on regardera comment elle est utilisée par la suite,
- **XDrawLine(...)** qui doit manifestement afficher l'ensemble des lignes du décor. On le trouve justement dans une boucle dont le test vérifie qu'une variable est bien inférieure à **nb_vertices**. Regardons-la de plus près :

```
for (o += (I = D * W + E * T * B,E* d / K *
B + v + B / K * F * D) * _;
     p < nb_vertices; )
{
  T = p[vertices_z] + i;
  E = c - p[vertices_y];
  D = vertices_x[p] - L;
  K = D * m - B * T - H * E;
  if (p[vertices_x] + vertices_y[p] +
p[vertices_z] == 0 |
      K < fabs(W = T * r - I * E + D * P) |
      fabs(D = t * D + Z * T - a * E) > K)
    N = 1e4;
  else
  {
    q = W / K * 4E2 + 2e2;
    C = 2E2 + 4e2 / K * D;
    N - 1E4 && XDrawLine(display ,window,
gc, N ,U, q,C);
    N = q;
    U = C;
  }
  ++p;
}
```

La partie initialisation de la boucle **for** a été placée ici uniquement pour nous égarer. Elle devrait être avant et ne nous intéresse pas pour l'instant.

Le test de cette boucle porte sur la variable **p** qui est justement incrémentée à la fin. Hormis dans le corps de la boucle, cette variable n'apparaît que deux fois : la première fois dans un appel à **sprintf()** pour des raisons assez obscures et une autre fois dans les paramètres d'un appel à **select()**. La valeur de **p** n'est jamais vraiment utilisée et sa dernière utilisation ne sert vraiment qu'à le remettre à zéro. On peut donc déplacer cette affectation dans la partie initialisation de notre boucle, et le **++p** dans la partie incrémentation.

On a donc une boucle qui parcourt chacun des sommets du décor pour dessiner des lignes. Mais ces lignes seront dessinées dans le repère de la fenêtre, alors que les données stockées dans les tableaux ne changent pas et sont dans le repère du monde (du décor). On peut s'attendre à ce que chaque position soit transformée dans le repère de l'avion (une translation et une rotation normalement) avant d'être transformée dans le repère de la fenêtre (par une transformation perspective et une mise à l'échelle au moins).

Les trois premières lignes de la boucle ressemblent à une translation, même si Banks a pas mal brouillé les pistes :

- Il a opéré d'abord sur z, puis sur y, puis sur x, {contrairement à ce qu'une personne saine d'esprit ferait}
- Il écrit les accès aux éléments d'un tableau **tab** aussi bien **tab[i]** que **i[tab]** (même si c'est autorisé en C, car **tab[i] == *(tab+i) == *(i+tab) == i[tab]**, c'est tout de même *{un signe de perversité, ça n'a pas du tout aidé Banks pour la mise en forme du code ! }*
- Il réutilise des noms de variables qui ont une autre utilité à l'extérieur de la boucle (après vérification, on peut utiliser d'autres noms pour **T**, **E** et **D** à l'intérieur de la boucle),
- Il utilise le négatif de certaines valeurs, juste pour embrouiller...

Au vu de ces trois lignes, on peut penser que le triplet de variables **(L, c, i)** correspond à la position de l'avion, on peut alors les renommer **plane_x**, **plane_y** et **plane_z**. De même, les variables **(D, -E, T)** sont les coordonnées des éléments du décor translaté dans le repère de l'avion. On peut les renommer **x_trans**, **y_trans** et **z_trans**.

Les trois premières lignes peuvent donc être réécrites ainsi :

```
double x_trans = vertices_x[p] - plane_x;
double y_trans = vertices_y[p] - plane_y;
double z_trans = vertices_z[p] + plane_z;
```

L'auteur *{toujours pour essayer de nous amadouer}* nous avait indiqué que l'altitude (z) était négative dans les calculs.

On a ensuite une ligne de calcul assez complexe. Mais le test suivant contient deux affectations que l'on peut sortir avant le test. Notez que comme ces affectations doivent absolument se faire, l'auteur n'a pas pu utiliser l'opérateur **||**, mais a dû utiliser **|**. Si on sort ces affectations, on peut utiliser **||**, ce qui est plus classique. Notez que **D** est encore réutilisé, il faut donc faire attention à nos renommages ! *{encore de la méchanceté gratuite... }*

Regardons ce que deviennent ces trois lignes de calculs avec les nouveaux noms de variables et une petite réorganisation des additions :

```
W = P * x_trans + I * y_trans + r * z_trans;
D = t * x_trans + a * y_trans + Z * z_trans;
K = m * x_trans + H * y_trans - B * z_trans;
```

Les plus matheux d'entre vous auront reconnu une multiplication d'un vecteur par une matrice. Il y a donc fort à parier que **[P I r;t a Z;m H -B]** soit une matrice de rotation *{sauf bien entendu s'il s'agit encore d'une ruse de ce détraqué...}*. On ne risque rien à renommer ces variables en **matrix[0][0]**, **matrix[0][1]** ... **matrix[2][2]**, si ce n'est évidemment que chacune de ces variables est réutilisée pour autre chose. Il ne faut donc renommer qu'entre

l'affectation précédente et l'affectation suivante (exclue !). Dans le même ordre d'idée, le triplet `(W D K)` représente le point du décor dans le repère orienté de l'avion (vu par le pilote, quoi). On peut donc renommer localement ces variables en `x_rotated`, `y_rotated` et `z_rotated`.

Le test suivant vérifie trois choses :

- que la somme des coordonnées du point dans le repère du décor ne soit pas nulle (ce qui n'arrive vraiment que si les trois coordonnées sont nulles en virgule flottante),
- que la valeur absolue de la coordonnée x dans le repère orienté de l'avion soit inférieure à la coordonnée z dans le même repère,
- que la valeur absolue de la coordonnée y dans le repère orienté de l'avion soit inférieure à la coordonnée z dans le même repère.

La première condition permet de créer des discontinuités dans le décor. Les deux autres vérifient que le point est bien dans la zone visible à l'écran.

Dans le cas où l'un de ces tests n'est pas réalisé, la variable `N` est mise à 10000 (certainement pour « marquer » ce point comme ne devant pas être tracé. Sinon, les deux variables `q` et `C` reçoivent le même genre de calcul : la coordonnée x ou y est divisée par la coordonnée z, ce qui permet de rendre compte de la perspective, en donnant une valeur entre -1 et 1. Cette valeur est ensuite multipliée par 400 (pour être (largement) mise à l'échelle de la fenêtre), avant de se voir ajouter la valeur 200 (pour être recentrée dans la fenêtre).

Il n'y a pas vraiment obfuscation dans cette partie. L'auteur a juste utilisé les notations 2e2 et 4e2 au lieu de 200 et 400, et n'a pas écrit les calculs dans l'ordre. *{Aurait-il eu une baisse de forme dans sa démence meurtrière ? }*

Par la suite, la fonction `XDrawLine()` est appelée pour tracer une ligne entre `(q, C)` et leur valeur précédente, mais uniquement si `N` est différent de 10000.

Si on renomme `(q, C)` en `(x_1, y_1)` et `(N, U)` en `(x_0, y_0)` (car ils contiennent les valeurs précédentes), on peut réécrire le bout de code concernant l'affichage ainsi :

```
for (p = 0; p < nb_vertices; p++)
{
  double x_trans = vertices_x[p] - pos_x;
  double y_trans = vertices_y[p] - pos_y;
  double z_trans = vertices_z[p] + pos_z;
  double x_rotated = matrix[0][0] * x_trans +
                     matrix[0][1] * y_trans +
                     matrix[0][2] * z_trans;
  double y_rotated = matrix[1][0] * x_trans +
                     matrix[1][1] * y_trans +
                     matrix[1][2] * z_trans;
  double z_rotated = matrix[2][0] * x_trans +
                     matrix[2][1] * y_trans -
                     matrix[2][2] * z_trans;
  if (vertices_x[p] + vertices_y[p] +
vertices_z[p] == 0 ||
      fabs(x_rotated) > z_rotated ||
      fabs(y_rotated) > z_rotated)
    x_0 = 10000;
  else
  {
    x_1 = x_rotated / z_rotated * 400 + 200;
    y_1 = y_rotated / z_rotated * 400 + 200;
    if (x_0 != 10000)
      XDrawLine(display, window, gc, x_0 ,y_0,
x_1, y_1);
    x_0 = x_1;
    y_0 = y_1;
  }
}
```

8 Affichage HUD pour garder la tête haute

Les lignes du décor ne sont pas les seules choses affichées par ce programme. Il y a aussi trois informations textuelles : la vitesse en nœuds, la direction et l'altitude.

Cet affichage est réalisé à l'aide des deux lignes suivantes :

```
sprintf(f, "%5d  %3d" "%7d",p = l /1.7, (y_1 =
9E3 + 0 * 57.3) % 0550,
       (int)pos_z);
...
XDrawString(display, window, gc, 20, 380, f, 17);
```

`f` est simplement un buffer dans lequel `sprintf()` va écrire et qui sera affiché par `XDrawString()`. Cette variable n'est pas du tout réutilisée ailleurs, on peut donc la renommer `text_buffer`.

Pour `"%5d %3d" "%7d"` qui est la même chose que `"%5d %3d%7d"` l'auteur *{ce fourbe !}* n'avait pas trop le choix ici. Il a dû couper la chaîne pour des raisons de mise en forme d'avion. On notera simplement que la chaîne de format indique 3 entiers.

Le premier est `p = l / 1.7` et doit être la vitesse en nœuds. On peut alors supposer que `l` est la vitesse en kilomètres par heure. Le `p =` n'est là que pour transformer le résultat en entier, puisque `p` est une variable entière ! On peut en profiter pour renommer `l` en `speed_z`. La vitesse a en effet a priori trois composantes et la vitesse « globale » de l'avion est vers les z positifs dans son repère (`speed_x` serait la vitesse vers la gauche (en cas de bourrasque de vent par exemple), et `speed_y` serait la vitesse verticale (donnée par la portance entre autres)).

Le deuxième entier est `(y_1 = 9E3 + 0 * 57.3) % 0550` et il doit représenter un angle donnant la direction de l'avion. Le 0550 est simplement un 360 en octal. Et 57.3 est 180/pi, permettant de transformer des radians en degrés. L'ajout de 9000 (qui est un multiple de 360) permet d'être sûr d'avoir un nombre positif (sauf si l'avion fait plus de 25 tours sur l'horizon...) et le `y_1` est là encore pour transformer un calcul flottant en entier. La variable `0` est donc l'angle donnant la direction de l'avion. C'est un angle que l'on appelle habituellement `yaw`. On peut donc réécrire cette partie ainsi : `((int)(9000 + yaw * 180.0 / M_PI)) % 360`.

Le troisième entier représente l'altitude et pour le coup, cela ne pouvait pas être moins obfusqué. *{Cela prouve que l'auteur sait faire du code lisible et qu'il a tout mis en œuvre pour me piéger.}*

La fonction `XDrawString()` affiche simplement le contenu de `text_buffer` à la position `(20, 380)` de la fenêtre, soit en bas à gauche.

Vous pouvez retrouver le programme avec toutes les modifications que l'on vient d'apporter sur mon site **[2]** (fichier **banks.graphic.c**)

9 Un avion, des angles, de l'air

On a réussi à avancer, mais il reste encore beaucoup de lignes assez obscures dans ce programme. On peut toutefois rendre le code plus clair ; les lignes du genre :

```
a += _ * (b = formule1, formule2)
```

en :

```
b = formule1;
a += (formule2) * dt;
```

dt est en effet plus parlant que **_** puisqu'il s'agit d'un delta de temps. Rappelons également que l'opérateur virgule ne fait qu'évaluer son premier membre, avant d'évaluer le second membre et de le renvoyer.

Une autre façon d'avancer un peu est de regrouper les lignes qui touchent les mêmes variables entre elles, mais l'idée *{démente}* de réutiliser les mêmes noms de variables pour différentes choses ne va pas simplifier cette tâche. Pour arriver à faire ça, j'utilise mon éditeur favori pour mettre en surbrillance tous les emplois d'une variable. Les lignes qui n'utilisent pas du tout une variable, et qui sont entre son affectation et son utilisation, peuvent être déplacées sans danger.

De plus, il faudrait arriver à comprendre comment la matrice **matrix** est construite et vérifier qu'il s'agit bien d'une matrice de rotation, sinon tout ce que je viens de dire pour l'affichage n'aurait aucun sens *{et nul doute que Banks s'en réjouirait ! }*.

L'idée est d'arriver à la structure de programme suivante :

```
Initialisation
Boucle
  Calcul de la matrice, mise à jour des angles
  Affichage
    Des lignes
    Du texte
  Gestion des touches
  Simulation, gestion de la physique, intégration
```

Et dans cette structure, nous avons déjà vu une bonne partie des blocs. Il nous reste à analyser le premier et le dernier bloc de la boucle.

Comme nous avons affaire à une boucle sans fin, on peut se permettre de déplacer certaines lignes de la fin de la boucle vers le début et réciproquement. Par exemple, les lignes **D = cos(o)** et **E = sin(o)** (en fin de boucle) peuvent être remontées au tout début de la boucle. En faisant ça, on s'aperçoit que les fonctions **sin()** et **cos()** ne sont utilisées que sur trois variables : **yaw** (on s'en doutait), **j**, et **o**. On peut alors penser que ces variables représentent des angles, et on remarque d'ailleurs rapidement que ces variables ne sont utilisées que pour être mises à jour et pour qu'on calcule leur sinus et cosinus. Malheureusement, ces calculs sont ensuite stockés dans les variables aux noms aussi peu parlants que **T** ou **W**. Comme **o** a une valeur nulle par défaut et que l'avion ne fait pas de tonneau par défaut, on peut supposer qu'il s'agit de l'angle « roll », et donc **j** serait alors l'angle « pitch », puisque l'avion gagne légèrement en altitude par défaut et que **j** est légèrement positif.

Je suppose ici que l'auteur a bien utilisé les angles d'Euler **[3]**, car c'est la façon classique de repérer un avion *{même s'il est bien capable d'avoir fait autrement...}*.

Si l'on renomme ces variables, ainsi que leur sinus/cosinus, en **sin_roll**, **cos_pitch**, etc., on obtient le code suivant pour la partie « calcul de la matrice, mise à jour des angles » :

```
double sin_yaw = sin(yaw);
double cos_yaw = cos(yaw);
double sin_roll = sin(roll);
double cos_roll = cos(roll);
double sin_pitch = sin(pitch);
double cos_pitch = cos(pitch);

matrix[0][0] = cos_yaw * sin_roll * sin_pitch - sin_yaw * cos_roll;
matrix[0][1] = cos_roll * cos_yaw + sin_roll * sin_yaw * sin_pitch;
matrix[0][2] = sin_roll * cos_pitch;
matrix[1][0] = sin_yaw * sin_roll + cos_roll * sin_pitch * cos_yaw;
matrix[1][1] = sin_pitch * sin_yaw * cos_roll - sin_roll * cos_yaw;
matrix[1][2] = cos_roll * cos_pitch;
matrix[2][0] = cos_pitch * cos_yaw;
matrix[2][1] = cos_pitch * sin_yaw;
matrix[2][2] = sin_pitch;

yaw += (cos_roll * F / cos_pitch + d / cos_pitch * sin_roll) * dt;
roll += (sin_roll*d*tan(pitch) + v + tan(pitch)*F*cos_roll) * dt;
pitch += (d * cos_roll - F * sin_roll) * dt;
```

On voit alors que la matrice n'est remplie que de formules de trigo (ce qui laisse bien à penser qu'il peut s'agir de rotation...). Et à y regarder de plus près, on reconnaît la forme des matrices que l'on voit dans la page **[3]**.

Le lecteur soucieux, matheux *{et aussi sociopathe que Banks}* pourra vérifier qu'il s'agit bien d'une matrice de rotation (orthonormale positive) en calculant les produits scalaires des vecteurs de cette matrice.

Les angles sont mis à jour par un calcul que je ne comprends pas complètement *{à la grande satisfaction de ce désaxé...}*, mais il est plus que probable que l'on ait un calcul d'une vitesse angulaire, multiplié par un quantum de temps (**dt**), pour avoir un delta d'angle qui est ensuite ajouté à chaque angle.

10 Un douloureux échec

Le calcul de la mise à jour des angles va peut-être s'éclairer avec les calculs de la partie simulation.

Les lignes commençant par **pos_** ont manifestement pour rôle de mettre à jour la position de l'avion dans le repère du décor (la position de l'avion dans le repère de l'avion ne change pas...). En nettoyant tout ça (réordonner les lignes et l'intérieur des lignes, on voit apparaître le code suivant :

```
pos_x += ( matrix[0][0]*M + matrix[1][0]*X + matrix[2][0]*speed_z)*dt;
pos_y += ( matrix[0][1]*M + matrix[1][1]*X + matrix[2][1]*speed_z)*dt;
pos_z += (-matrix[0][2]*M - matrix[1][2]*X + matrix[2][2]*speed_z)*dt;
```

On reconnaît à nouveau une multiplication de la matrice par un vecteur. En fait, il s'agit de la transposée de la matrice (on a inversé lignes et colonnes) et c'est donc la matrice inverse. **matrix**, comme utilisé dans le dessin du décor, permet de passer du repère du décor vers le repère de l'avion. Son inverse (comme utilisé ici) permet donc de passer du repère de l'avion vers le repère du décor. Le vecteur multiplié est **(M, X, speed_z)**. On peut donc supposer que **M** soit la vitesse latérale de l'avion (dans son repère) soit **speed_x** et **X** est donc **speed_y**. Ces trois lignes transforment donc le vecteur vitesse dans le repère du décor. La position de l'avion dans ce décor est donc modifiée de la valeur de cette vitesse multipliée par le quantum de temps.

Cool, on avance bien ! On s'aperçoit ensuite que la variable **T** contient le carré de la vitesse de l'avion (un produit scalaire du vecteur vitesse par lui-même). On devrait arriver à tout décortiquer facilement.

Sauf que ...

Non. Je n'arrive pas du tout à faire mieux. Je ne comprends rien du tout au reste du code qui restera pour moi une soupe de calculs abscons. Plein de constantes inexplicables. *{Me voilà à terre. Banks doit bien se moquer de moi. J'ai passé des journées (ou plutôt des nuits) à essayer de comprendre tout ça. Banks pousse l'humiliation jusqu'à expliquer qu'il a simulé un Piper-Cherokee. Je me suis donc documenté à ce sujet sur* **[4]**. *Mais je n'ai rien trouvé, pas la moindre explication de la moindre constante...}*

Mon psy essaie de me persuader que Banks n'a pas pu en 1998 prévoir que j'allais essayer d'expliquer son code à des centaines de millions de lecteurs et que ce n'est pas de la malveillance de sa part, ou que ce n'est pas vraiment contre moi. *{Je n'en suis pas sûr. Et je me demande s'ils ne se connaissent pas tous les deux pour rire de moi...}*

11 Version finale

Voici donc une version que j'ose tout de même appeler finale *{il faut bien que je me console et que je reprenne confiance en moi comme dit mon nouveau psy...}* :

```
#include <stdio.h>
#include <math.h>
#include <sys/time.h>
#include <X11/Xlib.h>
#include <X11/keysym.h>

#define dt 0.02

double vertices_x[999];
double vertices_y[999];
double vertices_z[999];
int nb_vertices;
double pos_x, pos_y, pos_z = 1000.0;
double matrix[3][3];
double speed_x = 0;
double speed_y = 7.26;
double speed_z = 221.0;
double yaw;
double roll;
double pitch = 0.033;

int x_0, y_0, x_1, y_1, p;
char text_buffer[52];

double P , T, D = 1, d, E, h = 8, I, J, K, m, t, u, v, W, S = 74.5, a, A = 32.2,
F, H;

int main()
{
  Display *display;
  Window root_window, window;
  GC gc;

  // Lecture des données 3D
  while (scanf("%lf%lf%lf", &vertices_x[nb_vertices], &vertices_y[nb_vertices],
&vertices_z[nb_vertices]) != EOF)
    nb_vertices++;
  // Initialisation graphique, création de la fenêtre, etc.
  display = XOpenDisplay(NULL); // Connexion au serveur
  root_window = RootWindow(display, 0);
  // Création d'un contexte graphique affichant en noir.
  gc = XCreateGC(display, root_window, 0, NULL);
  XSetForeground(display, gc, BlackPixel(display, 0));
  // Création d'une fenêtre de 400x400 pixels avec un fond blanc.
  window = XCreateSimpleWindow(display, root_window, 0, 0, 400, 400, 0, 0,
WhitePixel(display, 0));
  // Cette fenêtre réagira aux événements clavier
  XSelectInput(display, window, KeyPressMask);
  // Affichage de la fenêtre
  XMapWindow(display, window);
  for (; ;)
  {
    struct timeval G = { 0, dt * 1e6 };
    double sin_yaw = sin(yaw);
    double cos_yaw = cos(yaw);
    double sin_roll = sin(roll);
    double cos_roll = cos(roll);
    double sin_pitch = sin(pitch);
    double cos_pitch = cos(pitch);

    // Construction d'une matrice d'Euler (ZXZ modifiée)
    matrix[0][0] = cos_yaw * sin_roll * sin_pitch - sin_yaw * cos_roll;
    matrix[0][1] = cos_roll * cos_yaw + sin_roll * sin_yaw * sin_pitch;
    matrix[0][2] = sin_roll * cos_pitch;
    matrix[1][0] = sin_yaw * sin_roll + cos_roll * sin_pitch * cos_yaw;
    matrix[1][1] = sin_pitch * sin_yaw * cos_roll - sin_roll * cos_yaw;
    matrix[1][2] = cos_roll * cos_pitch;
    matrix[2][0] = cos_pitch * cos_yaw;
    matrix[2][1] = cos_pitch * sin_yaw;
```

```
    matrix[2][2] = sin_pitch;

    // Mise à jour des angles
    yaw += (cos_roll * F / cos_pitch + d / cos_pitch * sin_roll) * dt;
    roll += (sin_roll*d*tan(pitch) + v + tan(pitch)*F*cos_roll) * dt;
    pitch += (d * cos_roll - F * sin_roll) * dt;

    // Affichage des lignes
    x_0 = 10000;
    XClearWindow(display, window);
    for (p = 0; p < nb_vertices; p++)
    {
      double x_trans = vertices_x[p] - pos_x;
      double y_trans = vertices_y[p] - pos_y;
      double z_trans = vertices_z[p] + pos_z;
      double x_rotated = matrix[0][0] * x_trans + matrix[0][1] * y_trans +
matrix[0][2] * z_trans;
      double y_rotated = matrix[1][0] * x_trans + matrix[1][1] * y_trans +
matrix[1][2] * z_trans;
      double z_rotated = matrix[2][0] * x_trans + matrix[2][1] * y_trans -
matrix[2][2] * z_trans;
      if (vertices_x[p] + vertices_y[p] + vertices_z[p] == 0 ||
          fabs(x_rotated) > z_rotated ||
          fabs(y_rotated) > z_rotated)
        x_0 = 10000;
      else
      {
        // Caclul des coordonnées 2D
        x_1 = x_rotated / z_rotated * 400 + 200;
        y_1 = y_rotated / z_rotated * 400 + 200;
        if (x_0 != 10000)
          XDrawLine(display, window, gc, x_0 ,y_0, x_1, y_1);
        x_0 = x_1;
        y_0 = y_1;
      }
    }

    // Affichage du texte
    sprintf(text_buffer, "%5d  %3d" "%7d", (int)(speed_z / 1.7), ((int)(9000 +
yaw * 180.0 / M_PI)) % 360, (int)pos_z);
    XDrawString(display, window, gc, 20, 380, text_buffer, 17);

    // Gestion des touches
    while (XPending(display))
    {
      XEvent event;
      XNextEvent(display, &event);
      switch (XLookupKeysym(&event.xkey, 0))
      {
        case XK_Up: J++; break;
        case XK_Down: J--; break;
        case XK_Left: u++; break;
        case XK_Right: u--; break;
        case XK_Page_Up: h++; break;
        case XK_Page_Down: h--; break;
        case XK_Return: u = 0; break;
      }
    }

    // Mise à jour de la position de l'avion
    pos_x += (  matrix[0][0] * speed_x + matrix[1][0] * speed_y + matrix[2][0] *
speed_z) * dt;
    pos_y += (  matrix[0][1] * speed_x + matrix[1][1] * speed_y + matrix[2][1] *
speed_z) * dt;
    pos_z += (- matrix[0][2] * speed_x - matrix[1][2] * speed_y + matrix[2][2] *
speed_z) * dt;
    // Le code qui rend fou...
    T = speed_x * speed_x + speed_y * speed_y + speed_z * speed_z;
    I = speed_x / speed_z;
    D = v / speed_z * 15;
    m = 15 * F / speed_z;
    E = .1 + speed_y  * 4.9 / speed_z;
    t = T * m / 32 - I * T / 24;
    H = A * matrix[0][2] + v * speed_y  - F * speed_z + t / S;
    K = F * speed_x + (h * 1e4 / speed_z - (T + E * 5 * T * E) / 3e2) / S -
speed_y * d - matrix[2][2] * A;
    a = 2.63 / speed_z * d;
    speed_y += (d * speed_z - T / S * (.19 * E + a * .64 + J / 1e3) - speed_x *
v + A * matrix[1][2]) * dt;
    speed_z += K * dt;
    W = d;
    d += T *(.45 - 14 / speed_z * speed_y - a * 130 - J * .14) * dt / 125e2 + F
* dt * v;
    P = (T *(47 * I - m * 52 + E * 94 * D - t * .38 + u * .21 * E) / 1e2 + W *
179 * v) / 2312;
    v -= (W * F - T * (.63 * m - I * .086 + m * E * 19 - D * 25 - .11 * u) /
107e2) * dt;
    speed_x += H * dt;
    F += dt * P;
    select(p = 0, 0 , 0 , 0, &G);
  }
}
```

Cette version est évidemment sur mon site **[2]** sous le nom `banks.simu.c`.

11.1 Questions aux lecteurs

- J'ai triché sur quelques signes moins, saurez-vous les trouver et les expliquer ?
- La matrice de rotation est une variation des matrices d'Euler, voyez-vous pourquoi ?
- L'auteur de ce simulateur m'en veut-il vraiment personnellement ?

{- Connaissez-vous un bon psy ? J'en ai marre de me voir comme une merde ? }

12 La prochaine fois

La prochaine fois, j'essayerai de me refaire une santé morale avec un exemple un peu plus simple. Peut-être ce petit programme surprenant : `int main(){return 42;}` ■

Références

[1] http://ioccc.org/ : une source de codes illisibles.

[2] http://forma3dev.fr/vieux-barbu/ : mon site perso avec les fichiers sources.

[3] http://en.wikipedia.org/wiki/Euler_angles : une explication des angles d'Euler et les matrices associées.

[4] http://en.wikipedia.org/wiki/Piper_PA-28_Cherokee : la description de l'avion simulé. Ce lien n'est malheureusement pas vraiment utile.

ÉTUDIER L'EXÉCUTION D'UN SCRIPT PYTHON À L'AIDE DE SMILEY

par Tristan Colombo

Déboguer un script Python n'est pas chose aisée. Il existe bien le module pdb, mais celui-ci n'est pas forcément très simple à utiliser. Depuis peu, il existe une nouvelle alternative : Smiley.

Smiley n'est pas exactement un débogueur. Il est défini par son auteur Doug Hellmann comme un « traceur d'application », un outil pour espionner le fonctionnement des scripts Python et enregistrer leur activité. Outre le débogage, on peut donc utiliser cet outil pour analyser et comprendre le fonctionnement de certaines parties d'un code, ou encore réaliser des études de performances. Dans cet article, je vous propose d'installer et de tester le module **smiley**.

1 Installation

À l'heure où ces lignes sont écrites, **smiley** n'est disponible qu'en version 0.2. Le projet est certes très jeune, mais il a été initié par une figure du monde Python. Doug Hellmann a, entres autres, développé **virtualenvwrapper** pour simplifier l'utilisation des environnements virtuels et écrit la bible que tout développeur Python devrait posséder : « The Python Standard Library by Example » chez Addison Wesley.

Pour pouvoir tester ce module, il va bien entendu falloir l'installer. Nous allons utiliser le gestionnaire de paquetages **pip**, qui va aller chercher pour nous les modules nécessaires dans PyPi (*PYthon Packages Index*). Avant cela, pour vous épargner des recherches dues à divers paquetages manquants, il va falloir être certain d'utiliser **pip** pour Python 2.7 et posséder les entêtes Python :

```
~$ sudo aptitude install python-setuptools python-dev
~$ sudo easy_install pip
```

Comme vous risquez d'obtenir un message d'erreur pour un problème de gestion des dépendances sur le module **d2to1**, autant l'installer immédiatement :

```
~$ sudo pip install d2to1
```

Vous pouvez alors installer **smiley** (prévoyez environ une minute de compilation sur un ordinateur un peu puissant) :

```
~$ sudo pip install smiley
Downloading/unpacking smiley
  Downloading smiley-0.2.0.tar.gz
  Running setup.py egg_info for package smiley
 ...
  Installing smiley script to /usr/local/bin
Successfully installed smiley
Cleaning up...
```

Si vous avez strictement suivi cette procédure, vous ne devriez pas avoir de mauvaise surprise. En cas d'erreur dans la suite avec éventuellement un message « **unicode not allowed, use setsockopt_string** », vérifiez votre version de **pip** à l'aide de la commande :

```
~$ pip --version
pip 1.0 from /usr/lib/python2.7/dist-packages
(python 2.7)
```

Si, en fin de ligne, vous voyez apparaître **python 3.2**, c'est que vous n'avez pas suivi toute la procédure précédente, que **pip** était déjà installé, mais pas dans la bonne version. Vous devez réinstaller **easy_install** du paquetage **python-setuptools** et **pip**.

2 Première utilisation

Nous allons commencer par tester **smiley** sur un tout petit script que nous appellerons **test_1.py** :

```
01: print("Affichage d'un texte")
02:
03: # Variables
04: coeff = 3
05: liste = [1, 2, 3, 4, 5]
06:
07: # Opérations
08: for i in liste:
09:   print("{} x {} = {}".format(coeff, i,
i * coeff))
```

Ce code ne représente aucune difficulté, il s'agit simplement d'un code en Python 3.2.

Pour pouvoir l'étudier avec **smiley**, nous allons avoir besoin de deux consoles. Pour simplifier la lecture, les lignes de la première console seront préfixées par **(1)** et les lignes de la seconde seront préfixées par **(2)** :

- La première console va contenir la trace de l'exécution de notre code. Il faut exécuter :

```
(1) ~$ smiley monitor
```

Une fois le programme lancé, vous perdez la main sur la console qui

attend de recevoir des informations depuis le script exécuté dans une autre console.

- La seconde console va donc servir à exécuter le code à analyser :

```
(2) ~$ smiley run test_1.py
```

Sur la seconde console le code va se dérouler normalement :

```
(2) Affichage d'un texte
(2) 3 x 1 = 3
(2) 3 x 2 = 6
(2) 3 x 3 = 9
(2) 3 x 4 = 12
(2) 3 x 5 = 15
```

Sur la console d'analyse du code, nous allons pouvoir voir pas à pas l'exécution du script avec son nom de fichier, le numéro de ligne et l'état des variables locales. La sortie se fait de manière ininterrompue, jusqu'à la fin du script. Pour une meilleure lisibilité, je vais segmenter l'analyse des lignes obtenues :

```
(1) Starting new run: test_1.py
(1) test_1.py:   1: print("Affichage d'un texte")
(1) test_1.py:   1: print("Affichage d'un texte")
(1) test_1.py:   4: coeff = 3
(1) test_1.py:   5: liste = [1, 2, 3, 4, 5]
(1)                 coeff = 3
```

Au démarrage du script, nous avons bien l'affichage d'une chaîne de caractères par un **print** en ligne 1, puis la création de la variable **coeff** prenant la valeur **3** en ligne 4 et la création d'une liste de cinq entiers en ligne 5. Comme vous pouvez le remarquer, après l'affichage du contenu de la variable **liste**, nous retrouvons la valeur de la variable **coeff** : le contenu de toutes les variables locales est répété à chaque étape de manière à pouvoir observer leur évolution. Poursuivons l'analyse du code :

```
(1) test_1.py:   8: for i in liste:
(1)                 coeff = 3
(1)                 liste = [1, 2, 3, 4, 5]
(1) test_1.py:   9:   print("{} x {} = {}".format(coeff, i, i * coeff))
(1)                 coeff = 3
(1)                 i = 1
(1)                 liste = [1, 2, 3, 4, 5]
(1) test_1.py:   8: for i in liste:
(1)                 coeff = 3
(1)                 i = 1
(1)                 liste = [1, 2, 3, 4, 5]
(1) test_1.py:   9:   print("{} x {} = {}".format(coeff, i, i * coeff))
(1)                 coeff = 3
(1)                 i = 2
(1)                 liste = [1, 2, 3, 4, 5]
```

Ici, nous rentrons dans la boucle de la ligne 9, avec **coeff = 3** et **liste** qui contient toujours ses cinq entiers. À l'étape suivante, **i** va prendre la valeur de **liste[0]** et valoir **1** ; ce sera l'affichage de « 3 x 1 = 3 ». Les itérations vont se poursuivre et nous pourrons voir évoluer la valeur de **i**.

```
(1) ...
(1) test_1.py:   8: for i in liste:
(1)                 coeff = 3
(1)                 i = 5
(1)                 liste = [1, 2, 3, 4, 5]
(1) test_1.py:   8: return>>> None
(1) Finished run
```

Lorsque tous les éléments de la liste ont été parcourus, le script s'arrête et renvoie la valeur par défaut **None**.

3 Une utilisation plus complète

Maintenant que nous avons testé **smiley** sur un code simple, voyons quelles informations nous pouvons obtenir d'un code un peu plus compliqué faisant intervenir plusieurs fichiers et correspondant donc un peu plus aux programmes que l'on manipule dans la vie de tous les jours, tout en restant suffisamment succinct pour que les fichiers d'exemples restent clairs.

Le script **main.py** va importer les fonctions **disque()** et **boule()** du module **malib** que nous écrirons par la suite. Nous questionnons l'utilisateur pour obtenir un entier **n** qui servira de rayon et nous affichons l'aire du disque et le volume de la boule correspondant à **n** :

```
01: import malib
02:
03: if __name__ == "__main__":
04:   n = int(input("Donnez un entier : "))
05:
06:   print("Aire disque  : {}".format(malib.disque(n)))
07:   print("Volume boule : {}".format(malib.boule(n)))
```

Dans la ligne 4, la fonction **input()** renvoie une chaîne de caractères qu'il faut convertir à l'aide de **int()** avant de stocker le résultat dans la variable **n**.

Le fichier **malib.py** contient le code des deux fonctions de calcul et utilise le module **math** pour avoir accès à la valeur de Pi :

```
01: import math
02:
03: def disque(rayon):
04:   return math.pi * rayon ** 2
05:
06: def boule(rayon):
07:   return (4 / 3) * disque(rayon) * rayon
```

Nous pouvons lancer notre test en utilisant deux scénarii : dans le premier, l'utilisateur ne donnera pas un entier et une exception sera donc levée ; dans le second, tout se déroulera correctement.

3.1 La valeur saisie n'est pas un entier

Nous allons suivre la procédure classique et lancer **smiley monitor** dans un terminal et **smiley run main.py** dans un second terminal. Ici, il est intéressant de noter que l'analyse du code du premier terminal est mise en attente lors de la saisie de l'utilisateur :

```
(1) Starting new run: main.py
(1) main.py:   1: import malib
(1) main.py:   1: import malib
(1) malib.py:   1: import math
(1) malib.py:   1: import math
(1) malib.py:   3: def disque(rayon):
(1) malib.py:   6: def boule(rayon):
(1) malib.py:   6: return>>> None
(1) main.py:   3: if __name__ == "__main__":
(1) main.py:   4:     n = int(input("Donnez un entier : "))
```

Lors de l'import du module **malib**, nous pouvons voir que le nom du fichier d'exécution passe de **main.py** à **malib.py** avec le chargement des deux fonctions **disque()** et **boule()**. La dernière ligne indique que nous attendons la saisie de l'utilisateur.

Pendant ce temps, sur le second terminal, nous devons saisir une valeur :

```
(2) Donnez un entier : non
(2) Erreur : Entier attendu !
```

Le code s'arrête brusquement et du côté du terminal d'analyse également :

```
(1) main.py:   4:     n = int(input("Donnez un entier : "))
(1) 'message'
```

Nous savons donc que le problème a eu lieu en ligne 5 de **main.py**. Effectivement, l'utilisateur n'ayant pas saisi un entier, la conversion à l'aide de la fonction **int()** a échoué et a levé une exception. Nous allons corriger cela à l'aide d'un petit traitement par **try**/**except** et une boucle (les lignes en rouge sont les lignes qui ont été ajoutées) :

```
01: import malib
02:
03: if __name__ == "__main__":
04:   int_ok = False
05:   while not int_ok:
06:     try:
07:       n = int(input("Donnez un entier : "))
08:       int_ok = True
09:     except:
10:       print("Vous devez saisir un entier!")
11:
12:   print("Aire disque  : {}".format(malib.disque(n)))
13:   print("Volume boule : {}".format(malib.boule(n)))
```

Ce code est tout à fait valide et fonctionnera parfaitement avec un appel à la commande **python3 main.py**. Relançons nos terminaux smiley :

```
(1) Starting new run: main.py
(1) main.py:   1: import malib
(1) main.py:   1: import malib
(1) malib.py:   1: import math
(1) malib.py:   1: import math
(1) malib.py:   3: def disque(rayon):
(1) malib.py:   6: def boule(rayon):
(1) malib.py:   6: return>>> None
(1) main.py:   3: if __name__ == "__main__":
```

```
(1) main.py:   4:   int_ok = False
(1) main.py:   5:   while not int_ok:
(1)                 int_ok = False
(1) main.py:   6:     try:
(1)                 int_ok = False
(1) main.py:   7:       n = int(input("Donnez un entier : "))
(1)                 int_ok = False
```

Nous voyons apparaître les lignes qui ont été ajoutées et l'état de la variable **int_ok**. Comme précédemment, le programme attend la saisie de l'utilisateur et comme précédemment, nous ne saisissons pas un entier :

```
(1) main.py:   7:     n = int(input("Donnez un entier : "))
(1) 'message'
```

Et malheureusement, nous venons de mettre à jour une des limitations de cette version 0.2 : le terminal d'analyse est interrompu à cause de l'exception, alors que le terminal exécutant le programme a traité l'exception et rentre maintenant dans une boucle infinie. Pour tuer le processus, il va falloir le mettre en attente (CTRL+Z), récupérer son PID (identifiant de processus) et lui envoyer un signal de terminaison :

```
Donnez un entier : ^Z
[1]+  Stoppé                 smiley run main.py
~$ ps aux | grep smiley
tristan  15217  0.0  0.2 180660 17188 pts/7    Tl   21:38   0:00 /usr/bin/
python /usr/local/bin/smiley run main.py
tristan  15272  0.0  0.0  14616   916 pts/7    S+   21:44   0:00 grep smiley
~$ kill -9 15217
```

Voilà qui est bien ennuyeux... Mais que fait-on lorsque l'on rencontre ce cas de figure avec un logiciel libre ? On prend dix minutes pour créer un ticket et signaler l'erreur à la communauté (ou alors, avec plus de temps, on peut proposer un patch). Je me suis contenté de signaler le problème à Doug Hellmann dans le ticket #12 sur https://github.com/dhellmann/smiley. Pour l'instant, il n'a pas répondu, mais quand vous lirez ces lignes le problème aura peut-être été résolu.

3.2 Déroulement sans encombre

Passons maintenant au cas où nous saisissons une valeur correcte :

```
(2) Donnez un entier : 4
(2) Aire disque  : 50.2654824574
(2) Volume boule : 201.06192983
```

Sur le terminal d'analyse, nous nous étions arrêtés à la ligne 7 qui attendait la saisie d'un entier :

```
(1) main.py:   7:       n = int(input("Donnez un entier : "))
(1)                 int_ok = False
(1) main.py:   8:       int_ok = True
(1)                 int_ok = False
(1)                 n = 4
(1) main.py:   5:   while not int_ok:
(1)                 int_ok = True
(1)                 n = 4
(1) main.py:  12:   print("Aire disque  : {}".format(malib.disque(n)))
(1)                 int_ok = True
(1)                 n = 4
(1) malib.py:   3: def disque(rayon):
(1)                 rayon = 4
(1) malib.py:   4:   return math.pi * rayon ** 2
(1)                 rayon = 4
(1) malib.py:   4: return>>> 50.2654824574
(1) main.py:  13:   print("Volume boule : {}".format(malib.boule(n)))
(1)                 int_ok = True
(1)                 n = 4
(1) malib.py:   6: def boule(rayon):
(1)                 rayon = 4
(1) malib.py:   7:   return (4 / 3) * disque(rayon) * rayon
(1)                 rayon = 4
(1) malib.py:   3: def disque(rayon):
(1)                 rayon = 4
(1) malib.py:   4:   return math.pi * rayon ** 2
(1)                 rayon = 4
(1) malib.py:   4: return>>> 50.2654824574
(1) malib.py:   7: return>>> 201.06192983
(1) main.py:  13: return>>> None
(1) Finished run
```

L'entier étant correct, en ligne 8 la variable **int_ok** va passer à **True**. Notez que comme dans tout débogueur, smiley nous indique la ligne qui va être exécutée et le contenu des variables avant son exécution. C'est pour cela que l'on peut obtenir des séquences du type :

```
(1) main.py:   8:       int_ok = True
(1)                 int_ok = False
(1)                 n = 4
```

La variable **int_ok** vaut **False**, mais après exécution de la ligne 8, sa valeur sera bien sûr **True**.

La partie la plus intéressante de cet exemple se situe au niveau des appels de fonctions. Lorsque la fonction **boule()** de la ligne 13 est appelée, **smiley** nous signale que nous nous trouvons dans le fichier **main.py**. Nous rentrons alors dans le fichier **malib.py** (lignes 6 et 7) et, toujours dans ce fichier, nous passons à la ligne 3 pour aller rechercher la définition de la fonction **disque()**. Nous avons donc accès sans aucune manipulation à un classique pas à pas détaillé que l'on retrouve chez tous les débogueurs.

Conclusion

Même s'il n'est pas encore parfait, le module **smiley** a le mérite d'être très simple à utiliser. Pour le développeur novice en Python, il peut constituer un bon outil pour suivre le déroulement d'un script et comprendre comment corriger certaines erreurs. Pour les autres, **smiley** peut être utilisé comme un complément à **pdb**, une sorte de première passe permettant d'éliminer la majorité des bugs avant de passer à **pdb** pour les bugs les plus coriaces. ■

OPEN WORLD FORUM

Le premier forum libre et open source européen

BEFFROI
2, place Émile Cresp
92210 Montrouge

Mairie de Montrouge

openworldforum.org

DU 3 AU 5 OCTOBRE 2013

SUPPORT ET MAINTENANCE DE VOS LOGICIELS

OSSA

OPEN SOURCE SOFTWARE ASSURANCE

L'OPEN SOURCE ENFIN MAÎTRISÉ ET INDUSTRIALISÉ

CHOISIR LE SUPPORT OSSA C'EST :

- CHOISIR UNE OFFRE UNIQUE SUR LE MARCHÉ DE SUPPORT AVEC **ENGAGEMENT DE NIVEAU DE SERVICES (DÉLAIS ET RÉSULTATS) ET SANS LIMITE DE NOMBRE D'INSTANCES DÉPLOYÉES**.
- CHOISIR DE **MAÎTRISER LE CYCLE DE VIE DE VOS LOGICIELS EN PRODUCTION** AVEC UN SUPPORT PORTANT SUR LES VERSIONS QUE VOUS UTILISEZ RÉELLEMENT SANS QU'UN ÉDITEUR VOUS IMPOSE DES MONTÉES DE VERSIONS.
- CHOISIR UN **INTERLOCUTEUR UNIQUE** POUR PRENDRE EN CHARGE L'ENSEMBLE DE VOS LOGICIELS LIBRES EN PRODUCTION : UN EXPERT LINAGORA VOUS PREND EN CHARGE **24H/24 ET 7J/7** SUR LA PLATE-FORME 0800 LINUX POUR RÉSOUDRE VOS PROBLÈMES SUR PLUS DE 400 LOGICIELS LIBRES.
- CHOISIR UN VRAI **CONTRIBUTEUR** DU LIBRE : L'ENSEMBLE DE NOS CORRECTIONS ET DÉVELOPPEMENTS RÉALISÉS DANS LE CADRE DE LA MAINTENANCE CORRECTIVE ET ÉVOLUTIVE SONT INTÉGRALEMENT REVERSÉS AUX COMMUNAUTÉS DES LOGICIELS LIBRES, GAGE DE PÉRENNITÉ.
- CHOISIR UN CENTRE DE SERVICES QUI VISE **LE MAINTIEN EN CONDITIONS OPÉRATIONNELLES DES SOLUTIONS LIBRES UTILISÉES**, TOUT EN EXERÇANT UNE **VEILLE TECHNOLOGIQUE** ASSIDUE SUR LES LOGICIELS ORIGINAUX.
- CHOISIR UNE **ÉQUIPE D'EXPERTS PROFESSIONNELS ET IMPLIQUÉS DANS LES COMMUNAUTÉS** : LINAGORA COMPTE PARMI SES COLLABORATEURS LES CONTRIBUTEURS DE FEDORA, LEMONLDAP ::NG, MAGEIA, LIBREOFFICE.ORG, OPENSTACK, NETBSD, UBUNTU, DRUPAL, ROUNDCUBE, OBM, LINSHARE, LINPKI, PETALS ESB...
- CHOISIR UN SUPPORT INNOVANT 100% FRANÇAIS ASSURÉ DEPUIS LA FRANCE PAR LE LEADER HISTORIQUE DU LOGICIEL LIBRE AVEC LES MEILLEURS SPÉCIALISTES FRANÇAIS DE L'OPEN SOURCE.
- CHOISIR LE MÊME SUPPORT INNOVANT QUE DE GRANDS CLIENTS QUI ONT OPTÉ POUR LE **SUPPORT INNOVANT** : DGFIP, INSERM, GENDARMERIE NATIONALE, BANQUE DE FRANCE, ORANGE, SNCF, LA BANQUE POSTALE, MACIF, AIRFRANCE, ASSEMBLEE NATIONALE, GROUPE CASINO...

N'ATTENDEZ PAS POUR CHOISIR UN PROFESSIONNEL QUI S'ENGAGE RÉELLEMENT

TÉLÉCHARGEZ NOS LOGICIELS SUR WWW.LINAGORA.ORG

OBM.org — COMMUNICATION & COLLABORATION

LinShare — PARTAGE DE FICHIERS ET COFFRE-FORT ÉLECTRONIQUE

LinID — GESTION ET FÉDÉRATION DES IDENTITÉS, SSO

petals — ESB ET SOA

www.LINAGORA.com