GNU LINUX MAGAZINE / FRANCE

France Métro : 6,20€ - DOM 6,75€ - TOM 950 XPF - BEL : 6,80€ - LUX : 6,80€ - PORT. CONT. : 6,80€ - CH : 12,70CHF - CAN : 11,60$ - MAR : 70DH

L 19275 - 89 - F: 6,20 €

▶DÉCEMBRE ▶2006 ▶NUMÉRO 89

08 ▶KERNEL CORNER
Quoi de neuf dans Linux version 2.6.19 ?

14 ▶INTERVIEW
Le créateur de SYSLINUX répond à nos questions

18 ▶DESKTOP
e17 et les EFL, bien plus qu'un Desktop Shell

29 ▶XORG
Découvrez fr-oss, la nouvelle disposition du clavier français

38 ▶ANTI-SPAM
Tour d'horizon en pratique des solutions pour Postfix

70 ▶PERL
Bien utiliser les tests avec un langage permissif

76 ▶NOYAU
Développez vos pilotes de périphériques en mode caractère

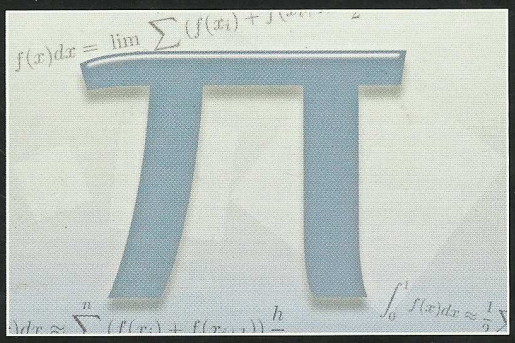
Utilisation avancée de Xen

43 ▶

Allez plus loin que la simple installation d'un moniteur de (para)-virtualisation. Apprenez à optimiser l'espace disque via LVM/snapshot et Copy-On-Write pour obtenir une solution fiable, disponible et sécurisée.

▶PROGRAMMATION CLUSTER

Calculer pi avec MPI



Via un exemple concret, découvrez le calcul parallèle et de développement d'applications cluster MPI reposant sur la bibliothèque MPICH

62 ▶

Administration et développement sur systèmes UNIX

EDITO

Ø4▶ DEBIAN CORNER
Ø4>Blobs et microcodes

Ø6▶ REFLEXIONS
Ø6>Les marques déposées et les Logiciels libres

Ø8▶ KERNEL CORNER
Ø8>Quoi de neuf dans Linux version 2.6.19 ?

14▶ PEOPLE
14>H.P.Anvin : M. ISOLINUX / SYSLINUX

18▶ UNIX/USER
18>Installation et configuration d'E17
29>Une nouvelle disposition de clavier français pour Xorg
32>Yafray, le moteur de rendu photoréaliste libre, une première approche

38▶ SYSADMIN
38>Lutter contre le spam avec Postfix
43>Xen et l'optimisation d'espace disque

48▶ HACKS/CODES
48>Manipulez avec les attributs

5Ø▶ DÉVELOPPEMENT
5Ø>Dissection de Glib : l'analyseur de ligne de commande
62>Programmation sur un cluster : calculer pi avec MPI
7Ø>Tests et Perl - Bonnes pratiques
76>Programmation noyau sous Linux Partie 2:pilotes en mode caractère
84>Le langage Ada - conteneurs
94>Smalltalk : un modèle pur objet

LÉGENDE :	
	> LIEN <
	> REMARQUE <
	> ATTENTION <
	> NOTE <
	> ASTUCE <
	> ANNEXE <

J'ai vu la fin de Google !

...et des autres moteurs de recherche basés sur des indices de popularité ou de la recherche textuelle. Cette vision m'est apparue subitement alors que je faisais un brin d'assistanat^Wassistance. Le cas était relativement simple. Le serveur dédié loué par l'utilisateur affichait des performances pitoyables. En effet, `hdparm -d1` retournait systématiquement `HDIO_SET_DMA failed: Permission denied`. Un cas classique d'utilisation du pilote générique IDE en lieu et place de celui dédié au contrôleur. Qu'à cela ne tienne, une reconfiguration du noyau suivie d'une recompilation règlera le problème.

Seulement voilà, `lspci` me signale un contrôleur intégré au *chipset* Intel 82801EB. Quel est le nom du support correspondant dans la configuration du noyau ? Demandons à Google... Et là, catastrophe ! La demande est simple, mais les résultats renvoyés sont affligeants, et vont du tuto iPod (`dmesg`) aux *posts* sur les forums façons « mon DMA, ça marche pô » en passant par les réponses constructives comme « chez moi non plus, aidez-moi ».

Le Web change et s'enrichit (sic) de contenu qui sont, le plus souvent, des demandes, voire des plaintes. Avec une recherche textuelle, on trouve donc de moins en moins facilement des réponses à nos questions. Même l'algorithme PageRank ne semble plus adapté dans ce type de situation.

Bien entendu, vous pouvez toujours utiliser les fonctionnalités avancées des moteurs de recherche et exclure les pages comportant les chaînes `"ça marche pas"` et `"moi non plus"`. Ca marchotte, mais il faut bien avouer qu'on en arrive à rechercher le motif de recherche...

C'est une évidence, un nouveau modèle de recherche devra être créé, car la technologie permet à de plus en plus de personnes de créer du contenu sur le Web. Historiquement, les experts furent les premiers à apporter leur contribution, mais aujourd'hui la répartition change et les réponses sont de plus en plus diluées dans le flot de questions, de commentaires sans intérêt et d'avis bloguesques... Un peu comme si votre fichier d'astuces et de commandes utiles se voyait complété aléatoirement et petit à petit par les TODO, les états d'âme et les interrogations de vos voisins.

Les solutions existent et ressemblent fortement aux techniques de lutte anti-spam, *scoring*, filtre bayésien... Les moteurs de recherche du futur seront ceux qui prendront ce virage, cela s'appelle la sélection naturelle.

Je vous retrouve le 30 janvier pour le numéro 90 et peut-être d'autres prophéties apocalyptiques ;)

Denis Bodor

P. S. : Pour la petite histoire, la version du noyau utilisée (2.6.18.1) nécessitait simplement l'activation du SATA dans le support IDE, en se passant de la libata donc. Le chipset, également appelé ICH5, n'était pleinement exploité qu'à cette condition.

GNU Linux Magazine
est édité par Diamond Editions
B.P. 20142 - 67603 Sélestat Cedex
Tél. : 03 88 58 02 08
Fax : 03 88 58 02 09
E-mail : lecteurs@gnulinuxmag.com
Service commercial : abo@gnulinuxmag.com
Sites : www.gnulinuxmag.com www.ed-diamond.com

Directeur de publication : Arnaud Metzler
Rédacteur en chef : Denis Bodor
Mise en page : Fabrice Krachenfels
Responsable publicité : Véronique Wilhelm Tél. : 03 88 58 02 08
Service abonnement : Tél. : 03 88 58 02 08
Relecture : Dominique Grosse

Impression : VPM Druck 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 61 72 76 24

IMPRIMÉ en Allemagne
PRINTED in Germany
Dépôt légal : A parution /N° ISSN : 1291-78 34
Commission Paritaire : 09 08 K78 976
Périodicité : Mensuel
Prix de vente : 6,20 €

www.gnulinuxmag.com

DEBIAN CORNER

→ Blobs et microcodes

Denis Bodor

EN DEUX MOTS Les blobs ne sont pas des monstres mutants dévoreurs de chair, mais la réalité n'est pas loin. Un blob est un objet binaire, et donc obscur, permettant d'utiliser un périphérique (souvent Wifi). Vous aurez compris le problème, un blob est un code binaire propriétaire utilisé via un wrapper GPL dans un noyau en GPL.

wodim
Graveur de CD

En réalité, ce paquet ne mérite pas vraiment de figurer ici, puisque tout le monde le connaît déjà... sous un autre nom. Il s'agit, en effet d'un *fork* de `cdrecord`, suite à certains problèmes techniques et sociaux. `wodim` est donc l'équivalent dans CDRkit (le fork) de `cdrecord` dans les CDRtools de Joerg Schilling.

Pour la petite histoire, les premiers problèmes entre Joerg et certains développeurs noyau datent de l'arrivée de Linux 2.6. Les développeurs ont décidé de réserver l'utilisation de certains ioctls au `root`, forçant ainsi l'utilisation du bit SUID ou plus simplement demandant à Joerg de modifier `cdrecord` de manière à ne plus utiliser ces ioctls.

Il existe un grand nombre d'entrées pour ce paquet dans le BTS. Pensez à suivre l'évolution des rapports/corrections régulièrement.

blktool
Tuning de disques

Vous connaissez sans doute l'outil `hdparm` permettant de configurer avec finesse les périphériques bloc ATA/IDE. `blktool` offre des services similaires, mais de manière plus générique, puisqu'il concerne à la fois la configuration de périphériques SCSI, IDE et SATA.

Ainsi, un `blktool /dev/hda dma on` est équivalent à un `hdparm -d1 /dev/hda`. Certains trouveront la syntaxe de `blktool` plus ergonomique, mais il lui manque encore quelques fonctionnalités pour devenir plus qu'un complément à `hdparm` (comme les mesures de performances par exemple).

Il n'existe pas d'entrée dans le BTS pour ce paquet. Ceci ne veut pas dire que le code est parfait. N'hésitez pas à faire un rapport en cas de bug avéré.

Les *blobs*, ou microcodes, représentent un point sensible pour toutes les distributions, car ils sont nécessaires pour la mise en œuvre de certains périphériques, mais entrent en contradiction avec les termes de la GPL et souvent les *guidelines* des distributions. Il en va de même pour les *firmwares* livrés avec certains pilotes. A l'instar des blobs, les firmwares sont des données binaires, mais celles-ci sont chargées dans le périphérique pour en permettre le fonctionnement, un peu à la manière des codes chargés dans la mémoire flash des microcontrôleurs.

Ces problématiques sont de plus en plus présentes dans les distributions GNU/Linux et le projet Debian est directement concerné. Le « contrat social » Debian impose un certain nombre de règles concernant l'intégration de logiciels dans une distribution. Il est dit clairement : « Nous reconnaissons que certains de nos utilisateurs demandent à pouvoir utiliser des travaux qui ne sont pas conformes aux principes du Logiciel libre selon Debian. Les paquets correspondants prennent place dans des sections nommées '*contrib*' (contributions) et '*non free*' (non libre). Les paquets de ces sections ne font pas partie du système Debian, bien qu'ils aient été configurés afin d'être utilisés avec lui. »

Si le noyau distribué par Debian inclut des blobs, microcodes ou firmwares, il devrait donc naturellement être déplacé dans non free. Mais, dans ce cas, comment supporter certains périphériques via l'installeur CD (comme les interfaces Ethernet `tg3`) ? La question se pose également s'il faut considérer l'intégration de versions « nettoyées » dans la distribution.

La résolution générale (GR) sur la question, la position officielle du projet, se fait attendre et la prise de décision est sans cesse reportée. Pour l'heure, l'équipe du noyau considère la solution suivante comme la plus viable. Elle distingue trois types de cas :

1. Les blobs binaires sans source, sans licence et sans autorisation de diffusion explicite : non redistribution et retrait (*prune*) des blobs en question du paquet `linux-*`.

2. Les blobs binaires sans source rediffusés sous GPL (blob + wrapper GPL) : c'est une violation évidente des termes de la licence GPL, mais leur suppression poserait un problème technique majeur, aurait un impact très négatif et contredirait l'un des objectif du projet consistant à propager le concept de Logiciel libre. La solution proposée est de tolérer ces pilotes pour Etch en espérant trouver une solution par la suite ; peut-être en amont, puisque les blobs sont clairement un problème pour les développeurs du noyau et Linus s'est clairement exprimé sur le sujet.

3. Les blobs binaires qui violent le contrat social pour d'autres raisons comme du code obfusqué ou avec des limitations concernant sa modification : les pilotes entrant dans cette catégorie sont moins nombreux mais, là encore, tolérer temporairement leur présence est un moindre mal.

Ce n'est toutefois pas la position officielle du projet, puisque aucune résolution générale (découlant d'un vote) n'a été annoncée.

Denis Bodor ,

db@ed-diamond.com
lefinnois@lefinnois.net

Les marques déposées et les Logiciels libres

Dave Neary

EN DEUX MOTS **Dans quelques semaines, une nouvelle version stable de Debian [1] sortira, probablement sans Mozilla Firefox. Voici pourquoi cet état de fait n'est pas si grave.**

Depuis quelques années, avec la popularisation grandissante des Logiciels libres, la cohabitation des marques déposées et les Logiciels libres devient de plus en plus difficile. Il me semble que beaucoup de développeurs de Libres ne comprennent pas la problématique des marques, surtout aux États-Unis.

Quand on est l'auteur d'une œuvre, le droit d'auteur est un droit passif – nous n'avons pas besoin d'affirmer notre droit, notre œuvre est automatiquement protégée par la loi. Par contre, le droit de protection des marques déposées est un droit que l'on est obligé de défendre si on souhaite conserver la protection de la loi.

Use it or lose it

Si j'apprends qu'un individu ou une société utilise une marque qui m'appartient, dans un domaine d'activité similaire (NDLR : la ou les classes dans lesquelles la marque a été déposée), et que je lui permette de continuer d'exercer son activité sans rien faire (peut-être parce que je pense que ce qu'il fait est « bien »), cela peut entraîner des problèmes par la suite avec des personnes malintentionnées. A l'instar d'O'Reilly and Associates et la marque Web 2.0, il faut normalement agir d'abord, puis discuter une fois que l'on a marqué notre territoire **[2]**. En effet, quelqu'un de malveillant pourrait démontrer que la marque a été abandonnée au domaine public, puisqu'on n'en parle pas et qu'on ne la défend jamais.

Le problème consiste donc à garder la marque « vivante » et défendue sans pour autant devenir totalement restrictif. Mozilla ou plutôt les avocats du projet Mozilla, comme la plupart des projets qui ont déposé des marques, a passé beaucoup du temps à rédiger des documents décrivant ce qu'il est possible ou non de faire avec leurs marques sans demander leur accord **[3]** (NDLR : ceci est également valide pour le logo puisque, en France par exemple, le logo ou sigle est déposé en même temps que la marque elle-même dans une ou plusieurs classes). Le projet négocie également des accords avec des tiers sur l'utilisation des marques en question lorsque les termes sortent du cadre de ces « guidelines ».

Une marque déposée est censée être un gage de qualité, qui assure l'origine du produit – comme une AOC pour les vins et les fromages. C'est un contrat de qualité et de confiance passé avec le consommateur. Cette confiance se gagne difficilement, et est très facile à perdre si une seule utilisation de la marque n'est pas à la hauteur des attentes.

Dans le cas des Logiciels libres, ce concept est en général inadapté, car de par sa nature, n'importe qui peut faire des changements et redistribuer le résultat de ses travaux sans la permission de l'auteur. L'appartenance du logiciel, l'AOC, ne peut donc pas être assurée.

Le cas Java

Java vient récemment d'être libéré sous une licence GPL v2, avec une exception Classpath pour une partie **[4]**. C'est une excellente nouvelle pour les développeurs de Logiciels libres. Mais la question des marques déposées se pose également chez Sun.

Selon Tim Bray **[5]** :

« Rassurez-vous : peu importe combien de *forks* de Java existent, ce ne sera pas Java à moins que cela ne s'appelle 'Java' ou que voyiez la tasse de café dessinée dessus. S'il y a le nom et la tasse, c'est Java, et c'est compatible. Et Sun protégera strictement cela devant les tribunaux si nous y sommes obligés. Nous l'avons déjà fait par le passé, et nous le referons. »

Comment font les distributions ?

Une distribution dispose de deux solutions. Si elle souhaite utiliser les marques de Mozilla par exemple, soit les responsables « remontent » toutes les modifications apportées au logiciel Mozilla afin de demander l'autorisation d'utiliser la marque du projet, soit cela rentre dans le cadre d'un accord qui leur donne le droit d'utiliser la marque, assurant qu'ils ne vont pas dépasser certaines limites. Dans le cas de Debian, un contrat n'est pas envisageable. Le *Debian Social Contract* **[6]** oblige les projets à donner à tous les utilisateurs les droits qu'ils donnent au projet Debian. Ceci est incompatible avec le concept visant à assurer l'origine et le niveau de qualité dans le contexte d'utilisation de marques déposées. La question n'est pas de savoir si Debian enfreint ou non les « *Mozilla trademark usage guidelines* », mais de simplement s'en tenir aux faits. Un développeur Debian précise qu'il « remonte » ses changements (patchs), mais Mozilla refuse de les valider **[7]**.

Un représentant de Mozilla dira que ces changements ne représentent pas de simples corrections de bogues, qu'ils sont trop conséquents et trop peu documentés pour être validés sans problème [8]. Au final, c'est l'impasse. Mozilla Corporation se sent obligée de retirer les droits d'utilisation de la marque au projet Debian, et au moment où j'écris ceci, il n'y a toujours pas de compromis à l'horizon.

Pourquoi, finalement, ce n'est pas grave

Quelle est la différence entre la Mona Lisa et la Joconde ? Si je suis à Bordeaux, je mets mes achats dans une poche, est-ce plus efficace que d'utiliser un sac à Lyon (NDLR : ou un sachet en Alsace) ? Est-ce qu'une salade de dent de lion est meilleure qu'une salade de pissenlit ?

Ce que Debian livre avec Sarge sous le nom d'IceWeasel sera le même logiciel qui allait y être intégré sous le nom de Firefox. Finalement, le nom ne change rien.

Je comprends que toutes les distributions souhaitent s'appuyer sur le succès du projet Mozilla, mais Debian ne sera pas meilleure avec le panda rouge qu'avec, exactement, la même chose sous un autre nom.

Le « droit au fork » est un principe important du Logiciel libre et, heureusement, Mozilla Firefox est un Logiciel libre. Si on n'est pas d'accord avec la façon dont un projet est géré, rien ne nous empêche de faire développer le logiciel sous un autre nom.

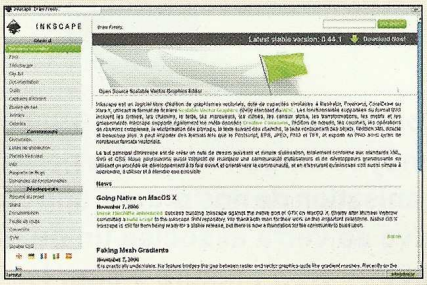
Heureusement, Sodipodi [9] était un Logiciel libre, et il nous a permis d'avoir Inkscape [10]. Heureusement la licence Mozilla [11] permet la création de projets comme IceWeasel... et Nvu [12]... et Epiphany [13].

Conclusion

Quel que soit son nom, Debian intégrera un *browser* Web performant, respectueux des standards Web, et génial dans son utilisation. C'est grâce aux Logiciels libres. Que demander de plus ? Soyons heureux !

LIENS

- [1] http://www.debian.org
- [2] http://radar.oreilly.com/archives/2006/05/web_20_service_mark_controvers.html
- [3] http://www.mozilla.org/foundation/trademarks/policy.html
- [4] http://www.sun.com/software/opensource/java/
- [5] http://www.tbray.org/ongoing/When/200x/2006/11/12/OSS-Java
- [6] http://www.debian.org/social_contract
- [7] http://web.glandium.org/blog/?p=99
- [8] http://cbeard.typepad.com/mozilla/2006/10/mozilla_tradema.html
- [9] http://www.sodipodi.com/index.php3
- [10] http://www.inkscape.org/



- [11] http://www.mozilla.org/MPL/boilerplate-1.1/mpl-tri-license-txt
- [12] http://www.nvu.com/index.php
- [13] http://www.gnome.org/projects/epiphany/

Quoi de neuf dans Linux version 2.6.19 ?

Éric Lacombe, Matthieu Barthélemy

EN DEUX MOTS Nous revoilà pour cette cinquième édition qui est intégralement consacrée à la version 2.6.19 du noyau Linux. Avec cette nouvelle mouture, nous faisons un peu de cosmétique pour parfaire l'organisation de nos propos afin que vous puissiez vous y retrouver plus facilement ;) Ainsi, chaque brève se verra associée un ou plusieurs tags facilitant le repérage. En voici une liste non exhaustive : « New Feature » pour les brèves développant une nouvelle fonctionnalité, « Technical » pour les brèves techniques détaillant un point particulier du noyau, « Kernel Addict » pour les apprentis sorciers ;) souhaitant mettre la main dans le cambouis, « Kernel Beginner » pour les brèves sur les aspects pratiques concernant la participation au développement noyau, « People » pour la narration de certaines polémiques récentes s'étant déroulées sur la LKML, etc. Vous allez découvrir également avec ce numéro une amélioration de taille ;) La rubrique d'actualité (taguée, comme il se doit, [Actualité]) est dorénavant structurée en ensembles cohérents vous permettant de vous y retrouver plus facilement. Noël avant l'heure ? ;)

ACTUALITÉ

Le noyau 2.6.19

Matthieu Barthélemy, bonsouere@gmail.com

Les périphériques blocs

La série 2.6.19 introduit plusieurs changements majeurs. Le plus célèbre est sans doute la bibliothèque PATA (*Parallel* ATA) et les drivers associés. C'est un sous-système générique complétant la libATA, qui vise à remplacer à terme le vieillissant système ATA/IDE du noyau. Les périphériques PATA et SATA profitent ainsi de la même infrastructure. Ce sous-système apporte une gestion d'un plus grand nombre de *chipsets*, de meilleures performances, et un meilleur comportement en cas d'erreurs se produisant au niveau du périphérique (secteurs défectueux, panne matérielle). Concocté par Alan Cox, il repart sur des bases neuves et saines, mettant au placard la collection de *hacks* accumulés dans la pile ATA/IDE ; nous pouvons ainsi en attendre une fiabilité accrue. Une bonne partie des chipsets les plus fréquemment rencontrés sont gérés (cependant marqués expérimentaux), et vous êtes cordialement invité à tester le fonctionnement du vôtre avec la libATA. A cette occasion, l'entrée *Serial ATA (prod) and Parallel ATA (experimental) drivers* fait son apparition dans le menu *device drivers* de configuration du noyau. Elle accueille les différents drivers de chipsets SATA/PATA, qui étaient auparavant situés dans *SCSI low-level drivers*. Sélectionnez votre chipset dans la liste, compilez votre noyau, et n'oubliez pas que, passant désormais par la libATA, vos disques ATA/IDE qui étaient vus comme HDX suivront désormais la nomenclature des périphériques SCSI (SDX). Pensez à adapter votre *boot* (option `root=/dev/hdX` en particulier ;-)). Enfin, il est plus que recommandé de sauvegarder vos données importantes avant de rebooter.

Allez, même si ça n'a aucune valeur fiable, un petit *bench* stupide avec la version 2.6.19-rc4, mon disque Maxtor et le sous-système PATA :

```
# hdparm -tT /dev/sda
/dev/sda:
Timing cached reads:   2004 MB in  2.00 seconds =
1001.55 MB/sec
Timing buffered disk reads:  170 MB in  3.02 seconds
=  56.22 MB/sec
```

Retour au « vieux » sous-système IDE : pour les *buffered disk reads* (lecture directe du disque sans passer par aucun cache), les résultats sont moins constants : j'oscille entre 48 et 54 MB/sec.

Matthieu Barthélemy, bonsouere@gmail.com

Les systèmes de fichiers

La partie « systèmes de fichiers » n'est pas en reste, pas moins de trois petits nouveaux font leur entrée, à commencer par le GFS2 de Redhat. C'est un FS destiné aux *clusters*, sur le même principe qu'OCFS2 inclus à partir du 2.6.16 (cf. Kernel Corner 85) : c'est un système de fichiers auquel plusieurs nœuds peuvent accéder simultanément, dans le cadre d'architectures SAN par exemple. Un exemple type de son utilisation est la création de clusters de bases de données (fichiers de données partagés par toutes les instances du SGBD). Plus ardu à configurer que son « rival » OCFS2, il offre le support des ACL, des quotas, des *direct* I/O, est extensible en ligne... Vous vous demandez peut-être quelles différences fondamentales justifient l'utilisation d'un de ces systèmes de fichiers « clusterisés » par rapport à notre traditionnel NFS, ou inversement.

Matthieu Barthélemy, bonsouere@gmail.com

Premièrement, deux interlocuteurs NFS discutent par-dessus la couche TCP (ou UDP) pour accéder à des données finales qui seront récupérées en faisant appel aux fonctions d'un système de fichiers « physique » (ext, reiser, XFS...). Nous voyons donc ici l'empilement de couches à traverser. GFS2 ou OCFS2 sont, quant à eux, des FS « natifs », c'est-à-dire assurant eux-mêmes l'organisation et la gestion du disque ; par ailleurs, G(ou OC)FS2 sont très nettement taillés pour converser par le biais d'un protocole de bus (iSCSI, *Fiber Channel*) spécifiquement dédié aux transferts via des liaisons directes entre équipements, ce qui a une incidence positive sur les performances, mais négative sur le coût des équipements :-) et la flexibilité de leur déploiement. Deuxièmement, ces deux derniers offrent une gestion précise et fiable des verrous lors d'accès concurrents à un même fichier, tandis que la fiabilité de NFS dans ce domaine est aléatoire.

La deuxième nouveauté FS de ce noyau est l'intégration du eCryptFS d'IBM, présenté lors du Ottawa Symposium 2005, qui n'est pas un système de fichiers à proprement parler, mais une surcouche proposant le chiffrage des données lors de leur stockage physique. Une démo pratique vaut souvent mieux qu'un long discours, voyons donc ensemble comment chiffrer un fichier.

1. Munissez-vous au préalable d'un 2.6.19 tout frais, compilez-le avec le support « *cryptographic* API », *Cryptographics options -> AES cipher algorithms*, *Security Options -> Enable access keys retention* et, enfin, *Filesystems -> Miscellaneous filesystems -> eCryptfs*.

Récupérons, puis compilons les paquets `keyutils` (**http://people.redhat.com/~dhowells/keyutils/**), puis `ecryptfs-util-4` (**http://ecryptfs.sourceforge.net/**).

2. Créons un montage `eCryptFS` :

```
# mkdir /tmp/jesuischiffré  <! le dossier des contenus
chiffré doit être dédié à ecryptfs >
# mkdir /mnt/jesuisdéchiffré
# mount -t ecryptfs /tmp/jesuischiffré/ /mnt/
jesuisdéchiffré/
eCryptfs interactive mount menu
----------------------------
        1. Mount with passphrase
        4. Abort mount
Make selection: 1
        Mount-wide passphrase: <entrez une passphrase ici ! >
        Confirm passphrase:
        Using the default salt value
Will attempt mount with authentication token signature
[d653285fa968513d]
Press 'l' to list available ciphers; enter for the default
cipher [AES-128]:
eCryptfs will use cipher [aes]
Performed successful eCryptfs mount using signature
[d653285fa968513d].
```

3. Allons-y !

```
# echo "rdv au centre commercial a 10h, amène la
valise" > /tmp/jesuisdéchiffré/message.txt
```

Tant que notre montage est actif, nous pouvons lire notre message (ceci s'applique bien entendu à tout contenu autre que texte) chiffré :

```
$ cat /mnt/jesuisdéchiffré/message.txt
rdv au centre commercial a 10h, amène la valise
```

Démontons (`umount /mnt/jesuisdéchiffré`) : il ne nous reste que le fichier réellement écrit sur le disque, donc `/tmp/jesuischiffré/message.txt`. Le résultat d'un `cat` sur celui-ci étant vomitif, je vous laisse vérifier par vous-même que vous n'y retrouverez pas votre message. Le contenu déchiffré sera accessible en recréant le montage avec la même *passphrase*.

Le troisième et dernier de la liste n'est rien de moins que... ext4. Présenté dans votre Kernel Corner 86, il est voué à devenir le successeur d'ext3, et les changements visibles pour l'instant par rapport à ce dernier sont l'adressage des numéros de blocs disque sur 48 bits (gestion de partitions jusqu'à 16 To) et les *extents* (manière d'inscrire les gros fichiers sur des blocs contigus, ce qui simplifie leur description). C'est un chantier qui sera en mouvement pendant plusieurs mois et dont les fonctionnalités supplémentaires ne sont pas encore complètement définies.

Matthieu Barthélemy, bonsouere@gmail.com

L'embarqué

A partir de sa version 2.6.19, Linux sera capable de tourner sur une architecture supplémentaire : les microcontrôleurs de type Atmel AVR32. Ces processeurs sont de type RISC et se destinent à l'embarqué, visant une efficacité de traitement par cycle d'horloge améliorée, afin de fonctionner à des fréquences plus basses et une consommation d'énergie moindre. Du côté de l'existant, il faut noter de nombreuses améliorations pour les X86_64 (*hotplug* mémoire et CPU), la gestion de plusieurs nouvelles puces de type ARM, le support de `cpufreq` pour les G5.

Possibilité utile pour les systèmes embarqués disposant de faibles ressources mémoire, désormais le sous-système de gestion des périphériques de type bloc peut être entièrement désactivé ; ce qui autorise malgré tout l'utilisation de systèmes de fichiers comme NFS ou JFFS.

Matthieu Barthélemy, bonsouere@gmail.com

Les pilotes de périphériques

Suite à des discussions ayant eu lieu depuis cet été au sujet de la (perfectible) gestion du *suspend/resume* dans le noyau, Linus Torvalds introduit de nouvelles fonctions permettant de dialoguer plus finement avec le matériel lors des phases d'hibernation/réveil. Ainsi la structure `bus_type` s'enrichit de deux nouveaux membres :

```
- suspend_late
```

appelée lorsque le système est presque totalement stoppé (mode monoprocesseur, interruptions désactivées).

```
- resume_early
```

agissant très tôt au réveil, également en mode monoprocesseur et avant activation des interruptions.

Ce sont des bases qui vont permettre, une fois que les pilotes les implémenteront, un meilleur fonctionnement de l'hibernation. Signalons également des améliorations de la couche USB dans ce sens, et la création d'un mode *debug* pour le *suspend* des processus et périphériques, permettant de mieux analyser leurs éventuels problèmes de mise en veille/réveil.

Une option « *parallel PCI probing* » (`CONFIG_PCI_MULTITHREAD_PROBE`) voit le jour, autorisant la découverte de plusieurs périphériques de type PCI en simultané. Ceci accélère notablement le temps de boot sur les systèmes multiprocesseurs, mais pourrait avoir l'effet inverse si trop de périphériques s'initialisent en même temps : la capacité d'énergie demandée pourrait dépasser les possibilités d'alimentation du bus, ralentissant l'initialisation des matériels connectés.

Parmi les traditionnelles évolutions au niveau des pilotes de périphériques, notons plus particulièrement, pour ce cru 2.6.19 :

- le support du *Direct Rendering Manager* (DRM) pour les puces Intel 965G ;
- le support des chipsets Intel 965 Express ;
- la gestion des cartes tuner Hauppauge WinTV-HVR3000, Nova-T 500 et HVR 1300 et de nombreux autres matériels d'acquisition analogique ou DVB ;
- la mise à jour des *Wireless Extensions* en version 21 ; effectuée de mauvaise grâce par les développeurs noyau qui attendent la bascule vers une autre solution comme une gestion via Netlink au-dessus de la pile générique Devicescape (cf. KC85) ;
- la gestion du WPA via les Wireless Extensions pour les puces WiFi Prism54, vous permettant d'en bénéficier depuis des applications comme le *Network Manager*. Le scan passif est désormais possible avec les matériels de type ipw2200 : contrairement au scan classique dit « actif », toute autre utilisation simultanée de la carte wifi est impossible, mais il a l'avantage de rapporter des informations supplémentaires (comme l'IP de toutes les machines connectées à un réseau, y compris celle de l'*access-point*).
- le retrait d'un bon nombre de drivers OSS, le support ALSA de ces périphériques étant considéré suffisamment stable.

Éric Lacombe, tuxiko@free.fr

La gestion mémoire

Plusieurs fonctionnalités au niveau de la gestion de la mémoire ont été ajoutées à cette nouvelle mouture. Une première concerne la traque des pages salies (*dirty* : comprendre écrites en mémoire mais pas encore sur le disque) dans les régions mémoires partagées avec le droit d'écriture. Cette fonctionnalité de Peter Zijlstra est implémentée en protégeant en écriture les pages concernées lorsqu'elles sont propres (*clean* : c'est-à-dire que les données en mémoire sont synchronisées avec le disque). Lorsqu'un processus tente une écriture, la faute de page est interceptée. La page est alors positionnée dirty et autorisée en écriture (via son PTE). L'écriture peut donc s'effectuer à ce moment (i.e. après la gestion de l'exception). Lorsque l'écriture sur le disque est effectivement lancée, c'est-à-dire quand les conditions du *page write-back* sont remplies (par exemple que le nombre de pages salies atteint un seuil critique), les *dirty bits* des PTE sont effacés et la protection en écriture est réactivée. Pour effectuer ces dernières actions, le *reverse mapping* est bien entendu utilisé. La nouvelle fonction `page_mkclean()` (ajoutée à `mm/rmap.c`) effectue ce travail.

Cette fonctionnalité étant implémentée, elle permet de réguler le nombre de pages salies en agissant directement sur les « fauteurs de troubles » (les processus écrivains des mappings partagés) en réduisant leur enthousiasme ;) Cela est implémenté dans un deuxième patch qui ajoute en gros l'appel à la primitive `balance_dirty_pages()` dans le gestionnaire de fautes de page précédent (`do_wp_fault`). Ainsi, en mettant au ralenti les processus fautifs, le page write-back (en gros, l'écriture sur le disque depuis la mémoire des pages non synchronisées) peut se dérouler sans encombre, évitant les situations où il n'y a plus de mémoire disponible à cause d'un nombre trop important de pages salies attendant d'être écrites sur le disque. La congestion de ce trafic pour les processus concernés, au moment opportun, évite ce genre de surprise.

Mel Gorman a ajouté un mécanisme d'enregistrement de régions de mémoire actives indépendant du type d'architecture. Une suite de patchs a converti ensuite les architectures ppc, x86, x86_64 et ia64 (pour le moment) à l'utilisation de ce mécanisme générique, entraînant une diminution flagrante du code spécifique à chaque type d'architecture chargé avant d'effectuer cette tâche. Avant l'introduction de ce dispositif générique, chaque type d'architecture définissait des structures pour enregistrer où se trouvaient les zones de pages actives. Ensuite, le calcul de la taille de ces zones ainsi que des trous

Éric Lacombe, tuxiko@free.fr

s'effectuaient. Maintenant, cette dernière étape est faite de façon générique. L'enregistrement, quant à lui, s'effectue pour chaque type d'architecture avec `add_active_range()`, laquelle reçoit les descriptions génériques des différentes régions. Le calcul sur les dimensions s'effectue ensuite génériquement au sein de `free_area_init_nodes()`.

Martin Schwidefsky a intégré une chaîne de notification au gestionnaire de rupture de mémoire (OOMM : *out of memory manager/killer*). Celui-ci a pour habitude de tuer des processus trop consommateurs, en cas de pénurie de mémoire. La fonctionnalité ajoutée permet d'enregistrer des *callbacks* qui peuvent libérer de la mémoire dans de telles situations. Le OOMM retente alors l'allocation responsable de son propre lancement et reporte à plus tard son boulot sanguinaire ;) Le but de cette chaîne de notification est d'ajouter un filet de sûreté quand des réserves de mémoire sont utilisées (par exemple les *mempool*). Si ces dernières sont agrandies par leur gestionnaire et empêchent une allocation mémoire de se produire ailleurs, il est préférable de les réduire plutôt que de mettre fin aux jours des processus.

La mémoire au sein d'un nœud du système (lié à un bus mémoire) est découpée depuis toujours en zones : les fameuses `ZONE_DMA`, `ZONE_NORMAL` et `ZONE_HIGHMEM`. Cependant, pour certaines architectures, elles ne sont pas pertinentes. Pour des architectures autres que x86_64, la `ZONE_DMA32` n'a pas de sens par exemple, ou bien encore la `ZONE_HIGHMEM` n'est utilisée, entre autres, sur aucune architecture 64 bits (l'espace d'adressage est suffisamment grand pour mapper directement en mémoire l'ensemble de la mémoire physique). Des structures de données sont associées à chacune de ces zones occupant de la place en mémoire. Elles sont toujours présentes au grand complet y compris dans les configurations où certaines zones ne sont pas pertinentes (l'abstraction en zone étant générique, le mécanisme de gestion n'est pas spécifique au type d'architecture). C'est pourquoi un patch a fait son apparition pour permettre la désactivation des `ZONE_DMA32` et `ZONE_HIGHMEM` au moment de la compilation du noyau.

Notons également une modification de sémantique des primitives `kmap/kunmap` (ainsi que les versions atomiques) censées être utilisées pour mapper temporairement en mémoire des pages situées en mémoire haute (ces dernières ne sont pas mappées directement en mémoire à la différence par exemple des 896 premiers méga-octets sur x86). Pour certaines architectures, des problèmes de cohérence mémoire nécessitent de *flusher* les pages du cache entre l'appel de `kmap` et `kunmap`. Maintenant, le développeur de pilote n'a plus à se soucier de cela, car cette gestion fait à présent partie du rôle de `kmap/kunmap`.

Un nouveau cas est dorénavant traité dans le gestionnaire de fautes de page avec l'ajout de `do_no_pfn()` par Jes Sorensen. Cette fonction gère la situation où le mapping mémoire ayant provoqué la faute n'est pas associé à une `struct page` (descripteur de page défini pour chaque page de mémoire physique). Cela évite la création d'entrées fictives dans les tables de pages pour les régions de l'espace d'adressage n'étant pas associées à de la véritable mémoire (le driver MSPEC supportant les opérations en mémoire spéciale pour l'architecture SGI N2 l'utilise par exemple). Des situations existent dans lesquelles il n'est vraiment pas conseillé d'avoir une `struct page` associée au mapping, car cette dernière offre un accès à la page en question qui peut être effectué en parallèle de l'accès par le driver. Si l'un utilise, par exemple, un accès en mode cache et l'autre un accès direct, des problèmes de corruption peuvent se produire.

Une nouvelle primitive est dorénavant disponible pour les développeurs noyau. Il s'agit de `void *kmemdup(const void *src, size_t len, gfp_t gfp)` qui permet de dupliquer directement une région mémoire. Avant, il était usuel de réserver la zone avec `kmalloc()` suivi d'un `memcpy()` pour effectuer la duplication (en ayant pris soin de vérifier le code de retour de `kmalloc()`). Cette nouvelle primitive effectue donc les deux actions en évitant au programmeur de faire des erreurs dans la saisie des longueurs (une pour le `kmalloc()` et l'autre pour le `memcpy()`). Bien qu'à première vue cette fonctionnalité ne semble pas transcendante, elle provient de la constatation que de nombreux bugs ont été le résultat de ce type d'erreur. Cette primitive est donc à préférer à l'ancienne méthode. De plus, le code noyau s'en trouvera réduit ;)

Éric Lacombe, tuxiko@free.fr

La virtualisation

Du côté de la virtualisation, plusieurs travaux sont en cours. Les fonctionnalités de container sont intégrées progressivement. Nous trouvons notamment la séparation d'espaces de noms pour différents types d'information : le « nom » du système (structure `utsname`) ; les objets IPC (possibilité de rendre privées les IPC – sémaphores, mémoire partagée, files de messages – via l'utilisation d'espaces de noms différents – création des `ipc_namespace`, ajout de l'attribut `CLONE_NEWIPC` pour la primitive clone, support de la primitive `unshare()`) ; les identifiants des processus (définition d'objets décrivant des espaces de PID) ; etc.

Notons également des patchs préparant l'arrivée prochaine du support pour la paravirtualisation. Avant de clore ce paragraphe, mentionnons la suite de patchs (pas encore intégrée à la branche principale du noyau) de Avi Kivity implémentant `/dev/kvm` (KVM : *Kernel Virtual Machine*) pour la création de systèmes invités. Nous aurons l'occasion de reparler de cela dans les prochains KC.

Éric Lacombe, tuxiko@free.fr

Les interruptions et les exceptions

Une modification au niveau de l'API de la gestion d'interruption générique a été apportée par David Howells. Elle consiste en la suppression d'un paramètre dans la primitive `generic_handle_irq` appelée à chaque interruption. `__do_IRQ()` est également modifié de la sorte. Ce changement provient de la constatation que l'argument `struct pt_regs *` n'est en fait utilisé que par très peu de gestionnaires d'interruptions. En conséquence, l'enlever de la liste des paramètres permet de gagner de la place sur la pile noyau (pensez par exemple à la chaîne d'appels imbriqués lorsqu'un périphérique USB branché à un HUB lève une interruption, et ce paramètre peut même avoir à passer par quelques couches supplémentaires : pour arriver jusqu'au gestionnaire `sysrq`). Nous y gagnons également en performance, puisque le code permettant le passage du paramètre disparaît (l'architecture FRV gagnerait 20% en rapidité sur le chemin de sortie du traitement des IRQ).

Les gestionnaires d'interruptions, ayant cependant besoin de connaître l'état des registres, utilisent dorénavant `get_irq_regs()`. Les informations sur les registres sont stockées dorénavant dans une structure `struct pt_regs *` qui est placée dans une variable globale pour chaque CPU (via la per-CPU variable `__irq_regs`). Pour maintenir cette structure, la fonction `do_IRQ()` stocke le pointeur `pt_regs` dans la per-CPU variable et conserve l'ancien. A la fin du traitement, l'ancien pointeur est restauré.

Notons enfin que le patch modifie plus de 1000 fichiers ;) Cela n'est pas étonnant au vu du nombre de gestionnaires d'interruptions existant dans le noyau (environ 1800).

Éric Lacombe, tuxiko@free.fr

Opérations d'entrées/sorties

Les opérations sur les fichiers sont définies en respectant une interface précise. Chaque driver en rapport avec ces opérations (notamment les systèmes de fichiers) doit définir une structure `file_operations`. Nous y trouvions jusqu'à présent en plus des opérations classiques, des versions dérivées. Ainsi, il existait une version vectorisée des opérations de lecture et d'écriture `readv() / writev()` et une version asynchrone (i. e. ne bloquant pas) `aio_read() / aio_write()`. La version asynchrone se voit maintenant étendue en des versions vectorisées (ce qui était un manque). De plus, la modification prépare également à la factorisation de toutes les opérations asynchrones et vectorisées, rendant l'interface `file_operations` plus claire et concise.

Éric Lacombe, tuxiko@free.fr, Matthieu Barthélemy, bonsouere@gmail.com

La sécurité

Un nombre assez important de modifications concernant la sécurité a pris place dans le 2.6.19. Cependant, la majeure partie concerne uniquement SELinux. Nous trouvons notamment parmi ces changements l'extension du système de fichier tmpfs afin de supporter les ACL POSIX (*Access Control Lists*) et les BSD *secure level* supprimés des LSM (*Linux Security Module*).

Voyons, à présent, les modifications au sein de SELinux. Tout d'abord, la version maximale du type de politique de sécurité supportée est dorénavant la 21. Cette version autorise les transitions entre toutes les classes de sécurité et n'est plus limitée uniquement à celle des processus. Elle supporte néanmoins toujours l'ancien format. En outre, la version maximale supportée par le noyau peut être abaissée artificiellement afin de pallier les problèmes de gestion éprouvés en *userland*, lorsqu'il est installé une nouvelle version d'un noyau supportant une version trop récente du format de la politique de sécurité en regard de celle installée sur le système.

Un important changement apporte un raffinement au niveau des Associations de Sécurité (AS) d'IPSec afin de pouvoir les utiliser dans un environnement MLS (*Multi-Level Security*). L'approche actuelle pour effectuer l'étiquetage des AS pour les besoins de SELinux utilise une correspondance directe entre les règles XFRM de la politique de sécurité (XFRM est le *framework* implémenté dans Linux permettant d'empiler les destinations afin qu'IPSec supporte les deux versions d'IP : v4 et v6) et les AS. Cependant, dans les systèmes MLS actuels, une règle XFRM (concernant, par exemple, les niveaux de sécurité « secret » et « top secret ») peut avoir besoin d'être associée à plusieurs AS (toujours dans l'exemple : les AS affectées aux contextes de sécurité « secret » et « top secret »). C'est chose faite dans la version 2.6.19 de Linux.

Une autre amélioration permet lors de la négociation IKE (*Internet Key Exchange*) de prendre en compte le contexte de sécurité afin de créer une AS unique correspondante.

Enfin, une autre modification rend possible l'utilisation des catégories MLS dans le paramètre `context` lors des opérations de montages de systèmes de fichiers.

Connu pour offrir une pile réseau très complète et étoffée, Linux propose désormais une infrastructure supplémentaire, le *packet labelling*. C'est une couche générique de sécurité qui permet à des systèmes distants de se mettre en accord et de définir des autorisations lors de communications réseau ; pour ce faire, chaque paquet IP reçoit un label dans son en-tête, contenant des informations de

Éric Lacombe, tuxiko@free.fr,
Matthieu Barthélemy, bonsouere@gmail.com

sécurité, ce qui peut par exemple permettre à son destinataire de le refuser s'il tente de transmettre un type d'information non autorisé, d'autoriser sa consultation uniquement par les utilisateurs spécifiés... Cette couche doit être couplée avec une implémentation de protocole de sécurité ; pour l'instant, c'est la solution CIPSO, devenue standard de fait, car déjà implémentée dans d'autres systèmes (Solaris...) qui est fournie.

Une protection intéressante est enfin intégrée : le support de l'option `-fstack-protector` de GCC >= 4.2. Proposée lors du *Kernel summit 2005* avec une série d'autres patchs appelée « ExecShield », elle transpose à l'informatique la technique des mineurs des siècles passés, qui détectaient grâce à des canaris la présence de grisou dans de nouvelles galeries avant d'y entrer eux-mêmes :-) Le principe de cette technique appelée « *stack-smashing protection* » est de placer aux endroits stratégiques de la pile (juste avant l'adresse de retour d'une fonction par exemple) un **canari**, qui est en fait ici une valeur connue. Si un bug/code malveillant cherche à écrire plus loin que ce qui est réservé, il écrase le canari. Ceci sera remonté au noyau qui se considérera alors comme non fiable et préférera se terminer en *panic* plutôt que continuer.

Éric Lacombe, tuxiko@free.fr

La synchronisation dans le noyau

Le 2.6.19 voit l'apparition d'un nouveau mécanisme de synchronisation (enfin, pas si nouveau que cela) : le SRCU (*Sleepable Read Copy Update*). En fait, il s'agit d'un dérivé du système RCU, déjà présent dans les noyaux 2.6. Cette version rend possible la préemption des *threads* lecteurs (fonctionnalité attendue notamment par la branche *realtime* du noyau).

Expliquons, à présent, le fonctionnement et le rôle du mécanisme RCU. Cet outil a été introduit dans le noyau afin de permettre un accès concurrent efficace sur une structure de données lorsque de nombreux lecteurs et écrivains sont mis en jeu. RCU est une structure sans verrou adaptée aux situations où de nombreux lecteurs ont besoin d'un accès concurrent. Il permet contrairement aux *seqlocks* (autre technique utilisée dans ces situations, mais faisant intervenir des verrous) un accès concurrent également pour les écrivains. De plus, le fait qu'aucun verrou ne soit utilisé aboutit à un gain de performance important. En effet, la mise à jour des compteurs au sein des verrous (par exemple pour seqlock) entraîne beaucoup d'invalidation de cache entre les différents processeurs dans un système SMP, impactant évidemment les performances.

Cependant, il ne peut pas y avoir que des avantages. Pour pouvoir utiliser cette technique, deux conditions sont nécessaires. La structure à protéger doit être accédée uniquement à travers un pointeur et il n'est pas autorisé de dormir au sein d'une section critique. Vous l'aurez compris, le nouveau mécanisme SRCU nécessite seulement la première hypothèse.

Expliquons, à présent, la magie de RCU ;) Du côté lecteur, avant d'entreprendre une lecture, nous devons faire appel à `rcu_read_lock()` qui ne fait que désactiver la préemption. A la fin de la lecture, nous réactivons la préemption via `rcu_read_unlock()`. C'est au niveau de l'écrivain qu'une astuce va s'effectuer. Quand nous souhaitons écrire dans la structure protégée, nous effectuons auparavant une copie de cette structure (via le déréférencement du pointeur d'accès) et nous la modifions à la place de l'originale. Une fois l'opération effectuée, nous basculons les structures via la modification du pointeur d'accès. Cette dernière opération étant atomique, l'utilisation de verrous n'est pas nécessaire. Remarquons une subtilité, la suppression de l'ancienne version ne peut pas être effectuée directement après l'opération de permutation des pointeurs. En effet, un lecteur sur une autre CPU peut encore y faire référence. Donc, au lieu de la supprimer, l'écrivain enregistre un callback (implémentant la suppression de l'ancienne version) dans une chaîne de notification. Ce dernier sera exécuté par une *tasklet* lorsque le système saura qu'il n'y a plus de référence en cours à l'ancienne version. Pour cela, il regarde à chaque *tick* si toutes les CPU sont passées par un état calme (*quiescent state*). Si le cas se présente, la tasklet est déroulée entraînant la libération de l'ancienne structure.

La dernière brique nécessaire à la compréhension du mécanisme concerne justement les états calmes. Une CPU est considérée être passée dans un état calme dans trois situations : la CPU effectue un *switch* de processus, la CPU vient de passer en mode utilisateur (i. e. passage du `ring0` au `ring3`), ou la CPU exécute le thread oisif (*idle thread* de `PID 0`). Il est imposé au processus lecteur de faire appel à `rcu_read_unlock()` avant que l'une de ces trois situations ne se produise. Cela est assuré justement par l'hypothèse de départ interdisant le blocage en lecture.

Nous reviendrons dans un prochain KC sur le fonctionnement détaillé du SRCU qui, je n'en doute pas, est en train de vous tracasser ;)

Comme à l'accoutumée, nous terminons en vous signalant que cette synthèse n'est pas complète, et développe ce qui nous a semblé le plus marquant et/ou utile au plus grand nombre. N'hésitez pas à approfondir le sujet via vos amis LWN, Kernelnewbies, KernelTrap et LKML :-)

→ H.P.Anvin : M. ISOLINUX / SYSLINUX

Erwan Velu & Anne Nicolas

EN DEUX MOTS **Interview réalisée par Erwan Velu via mail le 14 septembre 2006, traduite de l'anglais, puis relue par Anne Nicolas.**

Q GLMF : *Bonjour Hans Peter Anvin. Pourriez-vous vous présenter en quelques mots aux lecteurs de GLMF ?*

R Hans Peter Anvin : Bonjour, mon nom est H. Peter Anvin. Je me suis impliqué dans Linux depuis 1992 ce qui fait de moi un des premiers contributeurs. Au fil des ans, j'ai fait pas mal de choses autour du noyau Linux, mais aussi dans l'environnement utilisateur (*userspace*). Je suis principalement connu pour le site kernel.org, mais aussi Syslinux, `klibc`. J'ai également commencé la tribu Linux chez Transmeta où Linus Torvalds a travaillé pendant quelques années. Je suis né et j'ai grandi en Suède, ma langue natale est donc le suédois.

Q GLMF : *Depuis 10 ans, vous êtes le leader d'un projet que chaque utilisateur Linux a utilisé au moins une fois, et ce, parfois sans même le savoir. Le projet SYSLINUX est principalement utilisé comme « bootloader » pour cédérom. Pourriez-vous nous expliquer ce qui vous a poussé à démarrer ce projet ?*

R HPA : J'ai démarré le projet SYSLINUX en 1994 après une installation foireuse. Lorsque j'ai débuté sur Linux, la seule « distribution » qui existait utilisait deux disquettes (*boot* et *root*) ; le reste du système vous le bricoliez vous-même. Et par-dessus tout, le seul système de fichier qui était supporté était « Minix » et ne permettait pas de dépasser les 64 Mo. Courant 1994, une compagnie nommée « Softlanding » avait développé une distribution nommée « SLS » et j'allais l'utiliser pour remplacer mon bricolage. C'était la première fois que j'avais une distribution Linux sur cédérom. Malheureusement, l'installation a échoué car leur disquette n'avait pas de pilote pour ma carte SCSI et donc il ne pouvait pas accéder au lecteur de cédérom. J'ai pensé que cela serait une bonne idée si quelqu'un pouvait remplacer le noyau présent sur la disquette (c'était bien avant que les PC puissent démarrer sur cédérom, mais aussi avant les modules) sans pour autant avoir besoin d'un système Linux pour le faire.

Q GLMF : *D'après vous, qu'est-ce qui fait que ce projet est devenu incontournable pour booter les cédéroms ?*

R HPA : En fait, le composant que tout le monde utilise pour les cédéroms *bootables* est appelé « ISOLINUX ». Celui-ci fut développé un peu plus tard. La raison pour laquelle tout le monde l'utilise aujourd'hui est qu'il est très facile de réaliser un cédérom bootable avec ISOLINUX et mkisofs. Aujourd'hui, bien entendu, il est bien testé et fonctionne plutôt bien sur la plupart des machines à l'exception des machines possédant des BIOS de très mauvaise qualité.

Q GLMF : *Pourquoi avez-vous choisi d'utiliser la licence GPL pour vos travaux ? Quelle a été la raison pour le rendre libre ?*

R HPA : A l'origine, j'ai utilisé la licence GPL, car c'était celle que Linux utilisait. Rendre ce projet libre, c'était surtout pour me simplifier la vie. Si j'avais essayé de faire de l'argent avec, jamais le projet n'aurait fonctionné. Ma motivation n'était pas de me faire de l'argent, mais plutôt de permettre aux gens d'avoir une installation de Linux moins pénible, surtout quand les choses vont mal. Ceci dit, tout le travail que j'ai fait n'a pas été soumis à la GPL. Selon la cible des utilisateurs, je pense qu'il y a toute une suite de licences adaptées. Selon les cas, mon travail est réalisé sous licence MIT, LGPL ou GPL ou même la « nouvelle licence BSD ». L'utilisation des licences MIT/BSD ou LGPL/GPL dépend largement de la réponse à la question suivante : « Serais-je fâché si quelqu'un me prenait mon logiciel et le commercialisait sans mon consentement ? ». La réponse à celle-ci est « non » pour la plupart des projets et « oui » pour quelques-uns.

Q GLMF : *Quel cursus universitaire avez-vous suivi ? Cela vous a-t-il été utile pour votre travail et vos projets ? Avez vous appris l'informatique/programmation par vous-même ?*

R HPA : J'ai étudié à l'université de Northwestern près de Chicago. J'ai des diplômes en informatique et en électronique.

J'ai d'abord appris à programmer lorsque j'avais 8 ans sur un micro-ordinateur 8-bit suédois appelé « ABC80 ». J'avais l'habitude de m'asseoir pendant les récréations pour écrire de l'assembleur à la main.

Si cela ne fait pas de moi un *geek*, je ne sais pas ce qui pourrait le faire. Pendant mon adolescence, j'ai également conçu des circuits électroniques. A propos, j'ai récemment recréé l'ABC80 sur un FPGA. C'était un projet très amusant.

Puis, je suis allé au lycée. J'étais déjà un développeur assez confirmé. Cependant, je n'avais pas une très bonne formation théorique. J'ai raté la plupart des cours d'introduction pour directement attaquer les choses théoriques compliquées ; c'était bien.

Je voulais avoir un doctorat à l'université de Northwestern, mais j'étais déjà très impliqué dans Linux, et j'ai réalisé que si je devais continuer dans la recherche, ce serait sur les systèmes d'exploitation ou sur les compilateurs. Malheureusement, aucun professeur ne traitait de ces sujets, j'ai donc décidé de quitter l'université.

Q GLMF : *En 1999, vous avez démarré un projet annexe nommé « PXELINUX ». Celui-ci va aussi devenir un projet incontournable pour ceux qui veulent utiliser PXE pour démarrer un ordinateur à travers le réseau. Quel type de besoin vous a poussé à démarrer ce projet et surtout comment est-il devenu aussi célèbre ?*

R HPA : En fait, en 1999, c'était vraiment la honte que les PC étaient à peu près les seuls ordinateurs qui ne pouvaient pas démarrer en utilisant le réseau ; vous deviez acheter une ROM spécifique à votre matériel et votre système d'exploitation. C'était vraiment le bazar. Intel a réalisé que c'était une raison pour laquelle les gens achetaient d'autres machines (plus spécifiquement des Sun SPARC). Ils ont donc utilisé leur force pour imposer un standard.

Malheureusement, le standard qu'ils ont imposé, PXE, pourrait être décrit comme dément. Tout ce qui peut être fait est mal fait. Par-dessus ça, le logiciel qui était fourni et qui était supposé faire fonctionner ce nouveau standard était tellement *buggé* que la plupart des fonctionnalités n'étaient pas utilisables.

Il y avait donc besoin d'un petit *bootloader* de second niveau (petit, parce que certaines piles PXE explosent si vous essayez de télécharger plus de 32 K) et qui procurerait toutes les fonctionnalités que l'on attend d'un bootloader. Comme j'avais déjà SYSLINUX qui était déjà petit (il était conçu pour des disquettes), j'ai réalisé que le même code source pourrait être utilisé, plutôt que de repartir de zéro. L'avantage principal était que toutes les fonctionnalités de SYSLINUX étaient disponibles dans PXELINUX.

Q GLMF : *Depuis la version 3.00, un nouveau programme nommé « EXTLINUX » est apparu, mais reste nettement moins connu et utilisé que les autres projets (isolinux/pxelinux). D'après vous, qu'est ce qui fait qu'il est moins connu et utilisé que GRUB ou LILO ?*

R HPA : J'aimerais bien le savoir. EXTLINUX fonctionne sur plus de configurations que GRUB, mais GRUB est disponible depuis plus longtemps et les distributions Linux sont très réticentes au changement.

Q GLMF : *Dans le processus de développement de {pxe|ext|iso|linux}, quelle est la part de contribution de la communauté ? Est-ce suffisant ? Si non, quel type de développeur recherchez-vous ?*

R HPA : J'ai écrit la plupart du code du projet SYSLINUX. Ceci est en partie dû au fait que je suis très difficile sur la qualité du code, mais aussi parce que le cœur du produit est écrit en assembleur (nécessaire pour avoir un résultat compact). Peu de personnes ont un bon niveau en assembleur et, souvent, lorsque je reçois des patchs brillants de la part de certains utilisateurs, ils sont trop risqués.

Plus récemment, j'ai ajouté à SYSLINUX un environnement en plein développement appelé « COM32 ». Cela va permettre de rendre plus facile l'ajout de fonctionnalités et permettre ainsi à la communauté de contribuer plus largement. J'envisage même de remplacer une partie du code écrite en assembleur par du C. C'est un projet coûteux en temps et je ne suis pas sûr de l'avoir.

Est-ce assez ? Par rapport au temps que j'ai passé sur SYSLINUX, c'est relativement limité. Je pense que ce n'est pas assez dans le sens où je ne peux pas faire tout ce que j'aimerais réaliser dans ce projet. J'aimerais tout particulièrement voir plus de personnes pour compléter la documentation. La documentation de SYSLINUX n'est pas en très bon état, et ce, même après 12 ans.

Q GLMF : *Combien de temps attribuez-vous à vos projets ? (par semaine ou par jour)*

R HPA : Cela varie largement. Je passe environ 20 heures par semaine à mes travaux autour de Linux, mais c'est définitivement compliqué de tout concilier dans sa vie, cela implique bien évidemment mon travail et ma famille.

Q GLMF : *Vous étiez le webmaster de kernel.org, qu'est ce qui vous a fait arrêter cette tâche ?*

R HPA : Je suis toujours webmaster. Cependant, j'ai maintenant une équipe de cinq volontaires y compris moi-même. Ces derniers temps, la plupart du travail technique sur kernel.org est réalisé par les autres membres. Pour ma part, je m'occupe du sponsoring et des aspects juridiques.

Q GLMF : *Est-ce que le travail que vous réalisez sur ces outils est lié à certains besoins que vous avez dans votre travail ?*

R HPA : Quelques-uns, oui.

Q GLMF : *Est-ce que votre société utilise votre travail ? Si oui, vous donne-t-elle du temps pour les développer ?*

R HPA : Je pense qu'il est certain que chaque société qui utilise Linux utilise des outils que j'ai réalisés. La plupart des sociétés où j'ai travaillé étaient plutôt favorables, mais le temps que l'on me laissait pour ces projets était très variable. Transmeta, par exemple, supportait ces activités, mais après un changement de direction, on m'a demandé d'arrêter mes activités sur Linux. J'ai donc décidé que quitter cette entreprise.

Q GLMF : *Est-ce que votre société aide au développement d'initiatives Opensource ?*

R HPA : Je suis désolé, je ne peux pas commenter les activités de mon entreprise.

Q GLMF : *Depuis 10 ans, vous offrez à la communauté open source des outils incontournables. Que nous préparez-vous pour la suite ?*

R HPA : Ca fait plutôt 14 ans et même plus si l'on compte quelques outils libres que j'ai écrits pour MS-DOS. Mon projet principal pour le moment est de faire rentrer `klibc` dans l'arbre de développement du noyau et aussi quelques développements sur SYSLINUX. SYSLINUX 3.30 aura un support graphique avancé : c'était une demande récurrente. J'ai aussi quelques projets complètement nouveaux que j'aimerais commencer prochainement, mais je ne peux pas en parler pour le moment.

Q GLMF : *Quelle question étiez-vous certain que je vous poserai et que je n'ai pas posée ?*

R HPA : Une chose sur laquelle j'aimerais que les gens se pose des questions est la « partie obscure » de l'*Open Source*. J'ai passé beaucoup de temps et malheureusement d'argent à travailler sur des projets open source, mais, malheureusement, certains ne respectent pas ça. Régulièrement, je reçois des critiques de la part des utilisateurs qui régulièrement me demandent de régler des problèmes qui leur sont spécifiques, et ce, sans contrepartie, bien entendu. Plusieurs fois, j'ai vraiment évalué la possibilité de tout arrêter parce que cela devenait trop pénible, heureusement ça s'est passé.

Si vous utilisez du logiciel open source, s'il vous plaît, prenez un peu de temps pour remercier les développeurs. Ce n'est pas la peine d'envoyer de l'argent ou du matériel, mais parfois un email sympa ou une carte postale, c'est bien, voire mieux. Un simple merci donne envie de continuer. Nous faisons ce travail parce que les utilisateurs apprécient ce travail. S'il n'y avait pas d'utilisateurs, tout cela ne serait que du temps perdu.

Q GLMF : *Je voudrais vous remercier d'avoir accepté de répondre à cette interview pour permettre aux lecteurs de mieux vous connaître.*

R HPA : Merci d'avoir fait cette interview. C'est toujours sympa de rencontrer les utilisateurs. Et après tout, s'il n'y avait pas d'utilisateurs, tout le monde open source serait inutile.

LIENS

- http://kernel.org
- http://syslinux.zytor.com/pxe.php
- http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf
- http://syslinux.zytor.com/
- http://syslinux.zytor.com/extlinux.php
- ftp://ftp.abc80.org/pub/fpga/abc80/

→ Installation et configuration d'E17

Jonathan Muller & Jean-Christophe Lauffer

EN DEUX MOTS **Alors qu'une guerre sévère fait rage entre Gnome et KDE qui se veulent être des environnements de bureau complets, Enlightenment se fait une place de plus en plus grande au sein de nos distributions, en proposant un desktop shell simple, performant, flexible et thématisable.**

1. Introduction

Enlightenment DR17 (*Development Release*), alias **E17**, est l'évolution naturelle du très connu Enlightenment 16. Ce projet est en développement depuis 2000, date de release de la version 16. Le but initial pour le développeur principal, alias Rasterman **[1]**, et de son équipe **[2]** a été de fournir un ensemble de bibliothèques dynamiques entièrement nouvelles, appelées **EFL** (*Enlightenment Foundation Libraries*) comme une brique sur laquelle un maximum de programmes, dont E lui-même, peuvent s'appuyer. L'optique suivie lors du développement des EFL est « simplicité de l'API » **[13]** (par exemple, Emotion, la bibliothèque de lecture multimédia permet d'écrire un lecteur de DVD en 17 lignes de code C **[3]**) et « optimisation » (Enlightenment tourne sans problème sur des configurations matérielles réduites, ce qui lui a permis d'être choisi par les développeurs de chez Sony comme gestionnaire de fenêtres par défaut de la distribution Linux proposée sur la prochaine PS3).

Enlightenment est un *desktop shell*, c'est-à-dire qu'il ne gère « que » l'affichage de fenêtres, mais il le fait bien ! Ainsi, ne vous attendez pas à voir apparaître des icônes sur le fond d'écran lorsque vous branchez votre clé USB (il est possible de gérer les montages à chaud dans votre distribution, mais, dans ce cas, ça n'est pas géré par Enlightenment). Enlightenment mise avant tout sur la rapidité et la simplicité. Les actions se font principalement via des menus contextuels comme dans Fluxbox : un clic droit sur le bureau vous donnera accès à la liste des applications favorites, tandis qu'un clic gauche vous donnera accès au menu global. Toujours dans l'optique de simplicité, il suffit d'approcher le curseur de la souris du bord de l'écran pour changer de bureau ! Comment vous procurer ce gestionnaire de fenêtres fabuleux ? Eh bien, suivez-nous, on vous montre le chemin !

Figure 1 : E17, sobre et efficace

Cet article a pour but de vous aider à :

- compiler et installer E17, les EFL et les applications construites avec ces bibliothèques ;
- configurer et prendre en main E17.

De plus, nous allons, par l'intermédiaire du processus d'installation, décrire l'utilité des bibliothèques et logiciels disponibles.

2. Installation

Afin qu'aucun lecteur ne soit lésé, nous allons vous décrire l'installation de E17 à partir du code source disponible sur le dépôt **CVS** du projet. Nous verrons vers la fin de l'article comment utiliser un script qui fait tout le travail à votre place, ainsi que les méthodes d'installation de paquets sous Ubuntu et Debian (cf. un peu plus loin), et via Emerge pour Gentoo.

Tout d'abord, nous commencerons par récupérer le code source complet de notre fameux gestionnaire de fenêtres. Pour cela, il vous faudra un client CVS (installé par défaut sur la majorité des distributions, sinon, nous vous laissons le soin de l'installer par les moyens fournis par votre distribution préférée).

E17 dispose de trois modules principaux sur le dépôt CVS :

- `e17` : répertoire principal du projet, dans lequel se trouvent trois sous-répertoires qui nous intéressent : `apps` (applications), `libs` (EFL), `proto` (applications et bibliothèques en développement actif) ;
- `misc` : répertoire avec quelques projets basés sur les EFL, indépendants de Enlightenment ;
- `e_modules` : répertoire des modules E17. Les modules sont équivalents aux *applets* dans **Gnome** ; c'est-à-dire des petits programmes servant à étendre les fonctionnalités de E.

Pour récupérer ces sources (en prenant exemple sur notre arborescence) :

```
cd ~/repositories/e17
cvs -qz3 -d:pserver:anonymous:@anoncvs.enlightenment.org:/var/cvs/e
co e17 misc e_modules
```

Cette manipulation va créer les trois répertoires de sources. Avant d'attaquer la procédure de compilation, nous allons mettre à jour quelques variables d'environnement en supposant que le répertoire d'installation soit *~/e17* :

```
PATH : export PATH=~/e17/bin/:$PATH
LD_LIBRARY_PATH : export LD_LIBRARY_PATH=~/e17/lib/:$LD_LIBRARY_PATH
```

Vous pouvez bien sûr ajouter ceci dans les fichiers de configuration de votre shell. Une fois cette manipulation faite, nous allons commencer par compiler les bibliothèques du projet, appelées « Enlightenment Foundation Libraries », alias EFL. Pour ce faire, allez dans le répertoire `e17/lib` des sources fraîchement récupérées et entrez successivement dans les répertoires listés ci-dessous pour y exécuter les commandes suivantes :

```
./autogen.sh --prefix=~/e17
make all install clean
```

ATTENTION

Veillez à respecter l'ordre d'apparition dans le tableau ci-dessous en raison des dépendances entre les bibliothèques. De plus, vous aurez très certainement d'autres problèmes de dépendances avec des bibliothèques ou logiciels tiers, nécessaires à la compilation de l'ensemble. Nous vous fournissons ci-dessous une liste que nous espérons des plus complètes :

```
automake1.9 autoconf libtool gettext pkg-config build-essential
flex bison byacc x11proto-core-dev libfreetype6-dev libjpeg62-dev
libpng12-dev libtiff4-dev libungif4-dev zlib1g-dev libbz2-dev
libid3tag0-dev libncurses5-dev libgtk1.2-dev libglu1-mesa-dev
libxrender-dev libfontconfig-dev libdirectfb-dev librsvg2-
dev libcurl3-dev libxdamage-dev libxprintutil-dev libxine-
dev libltdl3-dev libtag1-dev libsqlite3-dev libtagc0-dev
libsmbclient-dev libcdio-dev libglade2-dev
```

2.1 Les bibliothèques

Bibliothèque	Description
`edb`	Bibliothèque de gestion de données basée sur Berkeley-DB, permettant la gestion d'informations sous la forme clé/valeur. Il s'agit là d'une manière simple et efficace de gérer des informations comme les données de configuration d'une application par exemple.
`eet`	Bibliothèque permettant de stocker des données sous un format compressé, puis d'accéder à ces données sans en décompresser la totalité.
`imlib2`	Bibliothèque de manipulation d'image. C'est une des bibliothèques principales des EFL. Enlightenment 16 était basé sur la première version d'`imlib`, `imlib2` en est le successeur. Il ne s'agit toutefois pas d'une évolution d'`imlib`, mais d'une réécriture complète de la bibliothèque. `Imlib2` gère les formats d'images suivants, qu'il peut lire, si besoin est, dans des archives compressées au format `zlib` ou `bzip2`, ou même dans un tag `id3` : ▶ `bmp` ▶ `gif` ▶ `jpeg` ▶ `lbm` ▶ `png` ▶ `pnm` ▶ `tga` ▶ `tiff` ▶ `xpm`
`imlib2_loader`	Prise en charge de formats d'images supplémentaires par `imlib2` non inclus dans la bibliothèque officielle. Cette extension ajoute la gestion des formats suivants, qu'il peut lire, si besoin est, dans des archives compressées au format `eet`, ou dans une base de données : ▶ `ani` – curseur animé ▶ `ico` – icône ▶ `xcf` – format d'image de l'application TheGimp **[4]**
`evas`	Bibliothèque de gestion de canevas, tout ce qu'affiche une application écrite avec les EFL lui est dévolu. Evas, à l'instar d'`imlib2`, fait partie des bibliothèques principales des EFL, et permet entre autres, d'afficher des lignes, des rectangles, des images, du texte et de gérer les événements. Note : pour frimer un peu devant les copains et copines (rares sont celles à qui cela a plu ;-)), nous avons compilé `evas` avec les options suivantes : ▶ `--enable-directfb` : pouvoir lancer une application `evas` sans avoir de serveur X qui tourne (en mode *frame buffer*). ▶ `--enable-gl-x11` : activation de l'accélération 3D de votre carte.
`ecore`	`ecore` est une boîte à outils pour le développeur, c'est une couche d'abstraction qui permet un accès simple entre autres à la gestion des événements, des communications via *sockets* et des accès à X. `ecore` fait aussi partie des bibliothèques principales des EFL.
`epeg`	Bibliothèque permettant de réduire très rapidement (le plus rapidement du monde selon Rasterman) des images au format JPEG en vignettes.
`embryo`	Langage de script et machine virtuelle permettant d'ajouter des capacités au moteur de thèmes `edje`.
`edje`	`edje` est un moteur de thèmes, permettant de changer l'apparence des applications écrites avec les EFL, en découplant totalement l'affichage et l'application. `edje` dialogue avec l'application par l'intermédiaire de mots clefs. L'étude de `edje` sera faite dans un article ultérieur.
`epsilon`	Bibliothèque permettant la réduction d'images en vignettes, `epsilon` est conforme au standard de l'initiative freedesktop.org, et gère tous les formats d'images gérés par `imlib2` (contrairement à `epeg` qui ne gère que le format JPEG). `epsilon` peut toutefois utiliser directement `epeg` pour améliorer ses performances.
`esmart`	Ensemble d'objets intelligents pour evas.
`emotion`	Bibliothèque de gestion multimédia. Elle est à l'origine basée sur la `libxine`, mais peut maintenant utiliser `gstreamer`. Si l'on regarde de plus près la configuration de la bibliothèque *emotion*, on s'aperçoit qu'il y a un support avec la librairie *xine* **[5]**, mais aussi *Gstreamer* un excellent *framework* multimédia **[6]**. Donc, en installant les paquets nécessaires à *Gstreamer*, vous pourrez bénéficier de ce framework dans vos applications *emotion*.

exml	Bibliothèque de gestion de fichiers/flux XML.
ewl	*Enlightened Widget Toolkit*. Bibliothèque de *widgets* qui délègue de manière totalement transparente l'affichage à edje.
engrave	Bibliothèque permettant le parcours et la modification des fichiers edje (.edc) compris ou non dans des fichiers .eet.

NOTE

Chaque bibliothèque dispose d'au moins un programme de test et de divers outils. Ils sont pour le moment compilés en même temps que la bibliothèque (certains ont été extraits de la compilation – c'est le cas pour edje, ecore, evas, eet et embryo – il faut alors jeter un œil dans le répertoire e17/test pour les compiler à part, via un appel à make). En voilà une liste non exhaustive (nous vous laissons le soin de jeter un œil dans votre répertoire d'installation) :

- Imlib2 :
  - imlib2_test : test global d'imlib2 ;
  - imlib2_show : autre test d'imlib2 ;
  - imlib2_poly : test de dessin de polygones;
  - imlib2_bumpmap : test de rendu d'un *bump mapping* sur une image [7].
- Evas :
  - evas_gl_x11_test : test de l'utilisation d'evas en mode fenêtré avec rendu OpenGL ;
  - evas_software_x11_test : test de l'utilisation d'evas en mode fenêtré avec rendu logiciel.
- Ecore :
  - ecore_evas_test : test de l'imbrication d'evas et d'ecore ;
  - ecore_test : test d'utilisation d'ecore, gestion des évènements de la souris et d'un *timer*.
- Edje :
  - edje : visualiseur de fichiers .edj (extension de fichier binaire edje);
  - edje_cc : compilateur de fichier .edj à partir d'un fichier .edc (extension de fichier source edje) ;
  - edje_decc : décompilateur de fichiers .edj ;
- Esmart :
  - esmart_test : affichage d'une image sur un fond en fausse transparence.
- Emotion :
  - emotion_test (à lancer avec au moins deux vidéos en paramètres) : les vidéos étant de simples objets evas, il est possible de modifier leur canal alpha, et ainsi de les rendre plus ou moins transparentes !
- Ewl :
  - ewl_embed_test : test d'imbrication evas/ewl (cliquez sur « open ») ;
  - ewl_test : visualisation, codes et tutoriels sur les widgets d'ewl.

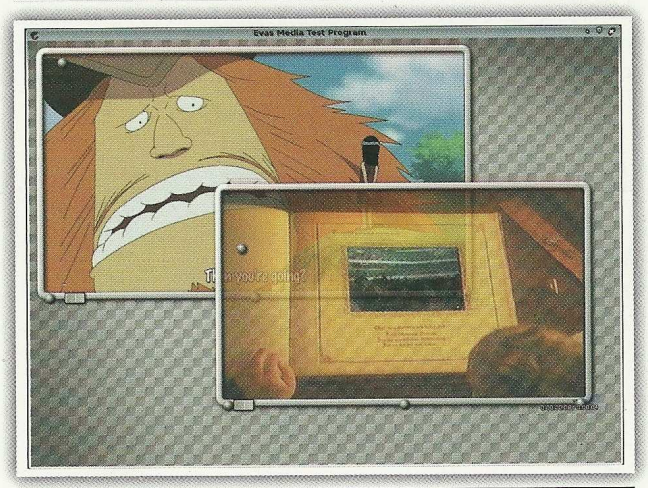


Figure 2 : emotion/evas/edje en action, jolie la transparence ;-)

2.2 Les applications

Ensuite, nous allons compiler les applications qui vont utiliser les bibliothèques construites ci-avant. Dirigez-vous dans le répertoire e17/apps de l'arborescence des sources. La compilation s'effectue avec les mêmes commandes que précédemment, toujours de préférence dans l'ordre d'apparition dans le tableau ci-dessous.

2.2.1 Le répertoire e17/apps

Ce répertoire contient les applications qui composent Enlightenment. Bien que la plupart soient encore en développement, elles sont considérées comme assez stables pour être diffusées.

Application	Description
e	E lui-même
e_utils	Outils de configuration de E
examine	Outil de configuration d'applications, il se connecte à une application active, et permet de configurer ses paramètres.
entice	Le visualiseur original d'images de E17, maintenant remplacé par exhibit.
eclair	Lecteur multimédia
elicit	Petit outil permettant entre autres de zoomer sur des parties de l'écran, ou de récupérer des couleurs.
evfs	Système de fichiers virtuel ajoutant une couche d'abstraction, ce qui permet d'accéder indifféremment à des fichiers de l'arborescence locale, d'un partage samba, ou encore d'une arborescence FTP.

entrance	Gestionnaire graphique de *login*. Note : l'installation d'entrance doit être faite en *root*, car les fichiers de configuration doivent être installés dans /etc.

Une application (autre que E) des plus représentatives de la puissance des EFL est certainement eclair. Comme énoncé dans le tableau ci-contre, eclair est un lecteur multimédia basé sur edje (le moteur de création de thèmes).

Figure 3 : eclair en action

NOTE

Il existe un petit utilitaire sympathique, compilé en même temps que eclair qui se nomme : eclair_wsz2edj. Ce petit utilitaire permet de convertir n'importe quel thème Winamp **[15]** en un *skin* pour eclair.

2.2.2 Le répertoire e17/proto

Ce répertoire contient des applications et bibliothèques considérées comme encore trop sujettes à modifications pour être intégrées directement à Enlightenment, mais certaines sont assez avancées pour que nous puissions vous proposer de les installer. Les commandes de compilation sont toujours les mêmes !

Application	Description
etk	Enlightenment ToolKit, cette bibliothèque est un *toolkit* qui a été développé à côté de ewl, de manière à être le plus proche possible de gtk. A l'origine conçu pour ajouter à eclair la gestion de playlists (fait en gtk pour le moment, ewl gérant très mal les dialogues de fichiers lors du développement d'eclair), etk est un toolkit très puissant, et beaucoup d'applications sont basées dessus, comme entropy ou exhibit.
elation	Gestionnaire de médias.
entropy	Explorateur de fichiers, basé sur la bibliothèque etk et evfs.
enterminus	Terminal écrit en EFL.
edje_viewer	Visualiseur avancé de fichiers edje.
ephoto	Gestionnaire d'albums photos.
exhibit	Gestionnaire d'images, écrit en etk.
enhance	Enhance est une bibliothèque qui permet de tirer profit des fichiers xml .glade générés par l'outil glade. Ces fichiers contiennent une définition d'interface destinée à un programme écrit en gtk. Grace à enhance, il est possible d'utiliser un fichier .glade pour obtenir une interface etk équivalente à celle définie en gtk. enhance ne produit pas de code, mais génère l'interface lors du lancement de l'application. Attention toutefois : enhance ne permet de générer que les widgets pour lesquels etk implémente un équivalent au widget gtk.
entrance_edit_gui	Application fenêtrée de configuration d'entrance. Note : à l'instar d'entrance, l'installation de cette application doit être faite en root, car les fichiers de configuration doivent être placés dans /etc.

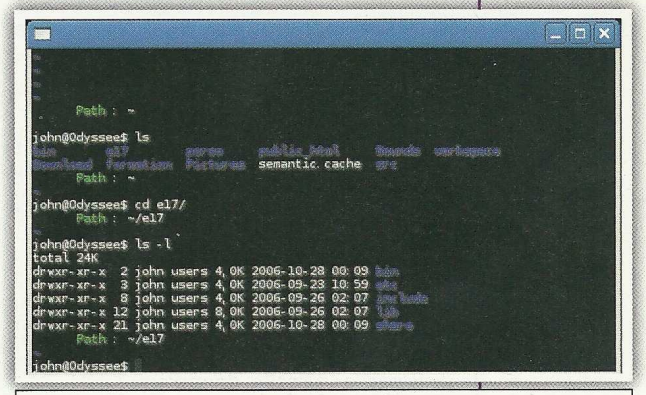


Figure 4 : enterminus, le shell basé sur les EFL

2.2.3 Les modules

Les modules sont des extensions de fonctionnalités d'enlightenment se trouvant dans le répertoire e_modules. Suivant l'état du dépôt CVS, certains peuvent ne pas compiler. Nous vous fournissons une liste ci-dessous, de modules dont le code est stable depuis quelques temps :

Module	Description
cpu	affichage de la fréquence du cpu
emu	Module éducatif, sert d'exemple pour le développement de modules
flame	Allume une animation brûlante en bas de votre écran
mail	surveille l'arrivée de courriels (pop, imap, mailbox...)
mem	Affiche l'utilisation mémoire
net	supervision des connexions réseau
rain	fait pleuvoir sur votre écran
screenshot	propose de faire des impressions d'écran
snow	fait neiger sur votre écran, sur des petits sapins...
tclock	affiche une horloge digitale
uptime	affiche le temps depuis lequel votre machine est allumée
weather	affiche la météo

Toute cette compilation devrait vous prendre environ 40-50 minutes. Il se peut que vous ne soyez pas motivé pour compiler tout cela à la main. La tâche peut paraître fastidieuse. Pour vous aider, il existe un script qui le fait pour vous ; son utilisation vous est expliquée ci-dessous.

2.3 Installation par un script

De nombreux scripts permettant une compilation et une installation aisées d'enlightenment ont fleuri un peu partout sur la toile. Nous allons ici utiliser le script officiel écrit par Morlenxus, un membre de l'équipe de développement d'enlightenment : `easy_e17.sh`.

Tout d'abord, récupérez le script :

```
wget http://omicron.homeip.net/projects/easy_e17/easy_e17.sh
```

Pour que la compilation se passe bien, il vous faut installer les mêmes dépendances que pour la compilation à la main (cf. chapitre 2), étant donné que ce sont les mêmes manipulations qui sont effectuées (elles vous sont juste cachées).

Le script n'a pas à être lancé en root, il suffit que sudo soit installé et que votre utilisateur ait le droit d'exécuter sudo. Le script détectera si vous êtes connecté en tant que root, ou vous demandera votre mot de passe pour sudo (utilisé pour l'installation).

Lancez le script, il vous fait tout (récupération des sources, compilation et installation... mais pas la vaisselle, dommage) !

```
./easy_e17.sh -i
```

Voilà, vous êtes parti pour 40-50 minutes, mais le résultat en vaut le coup ! Une fois le traitement terminé, déconnectez-vous, votre gestionnaire de connexion devrait vous proposer une nouvelle entrée dans la liste des sessions disponibles : enlightenment !

2.4 Dépôts Ubuntu et Debian

2.4.1 Debian x86 et x86_64

Ajoutez les dépôts binaires dans `/etc/apt/sources.list` :

```
deb http://edevelop.org/debian/ unstable main
```

Mettez à jour votre liste de paquets :

```
apt-get update
```

Il se peut que vous ayez un message d'information lors de la mise à jour des paquets concernant la clé non trouvée. Ce message n'est pas très grave, mais peut devenir pénible à la longue. Pour y remédier, copiez la chaîne de caractères de la clé et utilisez les commandes suivantes :

```
gpg --keyserver hkp://wwwkeys.eu.pgp.net --recv-keys la_cle
gpg --armor --export la_cle | apt-key add -
```

Enfin, une recherche du mot clé **e17** dans Synaptic ou un petit `apt-get install e17` devraient faire l'affaire. Déconnectez-vous de la session, une nouvelle session vous sera proposée dans gdm pour vous connecter à enlightenment !

Si vous souhaitez utiliser le gestionnaire de connexion `entrance`, installez-le via Apt_get ou Synaptic, le script d'installation vous demandera quel gestionnaire de connexion vous souhaitez utiliser : choisissez alors `entrance`.

Si vous souhaitez par la suite revenir en arrière, un petit `dpkg-reconfigure gdm` vous proposera le même choix.

2.4.2 Ubuntu Dapper

Grâce au travail de Lut!n et Sp4rky, les utilisateurs d'Ubuntu disposent d'un dépôt régulièrement mis à jour.

Dans un premier temps, téléchargez la clé de Lut!n pour authentifier le dépôt :

```
wget http://lut1n.ifrance.com/repo_key.asc
```

Puis installez-la (ce n'est pas obligatoire, mais ça vous évitera entre autres les messages de clé non trouvée lors de la mise à jour de la liste des paquets via Synaptic ou Apt-get) :

```
* sudo apt-key add repo_key.asc
```

Ajoutez les dépôts binaires et sources dans `/etc/apt/sources.list` :

```
deb http://edevelop.org/ubuntu dapper e17
deb-src http://edevelop.org/ubuntu dapper e17
```

Enfin, une recherche du mot clé « enlightenment » dans Synaptic, ou un petit `apt-get install enlightenment` devraient faire l'affaire. Déconnectez-vous de la session, une nouvelle session vous sera proposée dans gdm pour vous connecter à Enlightenment !

Si vous souhaitez utiliser le gestionnaire de connexion `entrance`, installez-le via Apt_get ou Synaptic, le script d'installation vous demandera quel gestionnaire de connexion vous souhaitez utiliser : choisissez alors `entrance`.

Si vous souhaitez par la suite revenir en arrière, un petit `dpkg-reconfigure gdm` (ou tout autre) vous proposera le même choix.

2.5 Gentoo et FreeBSD

Nous n'entrerons pas dans les détails de l'installation, mais des ports sont disponibles sous Gentoo et FreeBSD. N'hésitez pas à parcourir vos arborescences !

2.6 Autres distributions

Il se peut que des volontaires aient effectué un travail similaire pour votre distribution préférée, mais, dans ce cas, ces travaux n'ayant pas été recensés sur edevelop, nous ne vous en faisons pas part (par méconnaissance sans doute). Si c'est le cas, n'hésitez pas à communiquer ces initiatives auprès de la communauté Enlightenment !

3. Lancement de E et prise en main

Dans cette partie, nous allons voir comment lancer E de différentes manières et nous allons nous familiariser avec les outils que propose ce *desktop manager*.

3.1 Lancement de E

Il y a trois façons de lancer E :

- utiliser le gestionnaire de connexion `Entrance` ;
- configurer votre gestionnaire de connexion favori (gdm, kdm, xdm...) afin de rajouter une entrée pour E;
- utiliser la commande `startx` en mode console.

3.1.1 Entrance

Figure 5 : Le gestionnaire de connexion entrance

3.1.2 Configuration de gdm

Nous allons prendre comme exemple le gestionnaire de connexion du projet Gnome : gdm. Pour pouvoir lancer E via gdm, rendez-vous dans le répertoire : `/usr/share/xsessions/` (vous allez avoir besoin de votre compte root). Dans ce répertoire, éditez un fichier `Enlightenment.desktop` et recopiez ceci :

```
[Desktop Entry]
Encoding=UTF-8
Name=E17
Exec=/home/john/e17/bin/enlightenment #Chemin vers votre propre binaire.
Icon=
Type=Application
```

Relancez ensuite gdm et dans le choix de sessions, vous trouverez une nouvelle entrée E17.

3.1.3 Startx

Si vous ne possédez pas d'accès root ou que vous avez tout simplement la flemme d'éditer la configuration de votre gestionnaire de connexion, il vous suffit de créer un fichier à la racine de votre compte : `.xinitrc`. Ce fichier contiendra cette unique ligne

```
/home/john/e17/bin/enlightenment #Chemin vers votre propre binaire.
```

Il suffit alors de couper votre session X courante : [Ctrl]+[Alt]+[F1], puis en root d'exécuter la commande `/etc/init.d/gdm stop` (ou tout autre gestionnaire de connexion), ce qui aura pour effet d'arrêter l'interface graphique X. En tant que simple utilisateur, tapez ensuite `startx` pour la relancer directement, le fichier `.xinitrc` sera automatiquement pris en compte.

Une fois vos différentes manipulations faites, vous pouvez lancer E. Vous allez apercevoir une magnifique animation au ralenti. Les images ne vous donneront qu'un aperçu de la chose. Nous vous invitons à visionner une vidéo disponible sur le site de **Rasterman [8]**. Cette dernière n'est pas à jour avec les versions actuelles de E, mais elle donne un bon aperçu du rendu. Une fois E lancé, vous pouvez cliquer avec le bouton gauche de votre souris pour naviguer dans les panels de configuration (figure 6).

3.2 E_Utils

Si vous avez suivi la procédure d'installation, nous avons compilé des utilitaires tels que `e17setroot` qui permet de définir manuellement votre fond d'écran. Cette application est très utile si vous utilisez des logiciels manipulant la fausse transparence (ETerm, etc.).

```
$ e17setroot [options] monWallpaper.png
```

ATTENTION

Si vous lisez en parallèle de cet article d'autres tutoriels sur internet, il se peut que vous tombiez sur des termes qui sont désuets depuis le début du mois de septembre. Notamment, la notion de « eap » (pour *E Application*). Ce format était un format binaire afin de garantir les performances du bureau. Un binaire eap était un binaire de description d'un autre exécutable, un lien vers une autre application. Ce format a été remplacé par la norme de fichiers `.desktop` prodiguée par freedesktop.org **[9]** et utilisée dans Gnome, KDE...

3.3 Configuration des icônes

Il y a plusieurs façons de configurer les icônes présentes. Par l'intermédiaire du Menu *E* (clic gauche) -> *Configuration* -> *Applications*. Par ce chemin, vous arrivez sur un panneau de configuration, avec la liste des « bar » pouvant accepter des raccourcis.

Si les interfaces graphiques pour configurer votre système ne sont pas votre tasse de thé, il y a bien évidemment une alternative à ce panneau.

Pour pouvoir configurer le tout à la main. Il va falloir quelques informations supplémentaires. Le répertoire de configuration de E se trouve dans : `~/.e`. Les applications sont dans `~/.e/applications/all`. Dans ce dernier répertoire, vous aurez la liste de tous les fichiers à extension `.desktop`. Si vous voulez configurer votre module *ibar*, il vous suffit

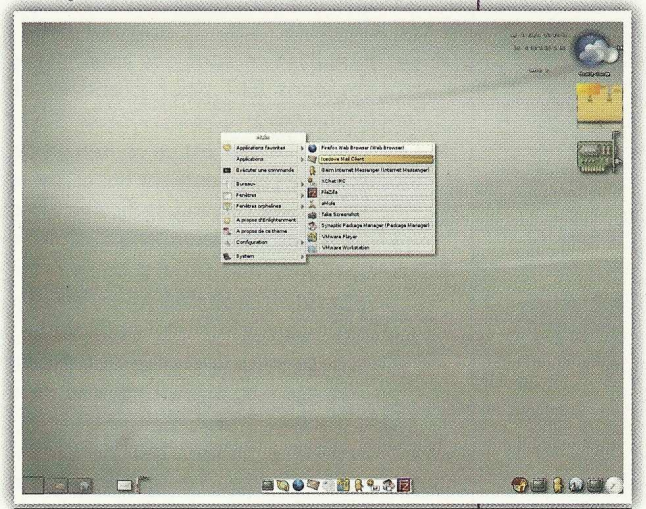


Figure 6 : Le bureau et le menu accessible via un clic sur le bouton gauche de la souris, plus quelques modules (de gauche à droite et de haut en bas : uptime, weather, mem, net, pager, mail, température, ibar, ibox, horloge).

de vous diriger vers :~/.e/e/applications/bar/default et de faire :

```
echo "monAppli.desktop" >> .order
```

NOTE

Le fichier .order est le fichier qui représente l'ordre d'affichage des icônes dans le cas d'une barre de lancement, et l'ordre de lancement des applications dans le cas du démarrage de E (~/.e/e/applications/startup).

Il se peut aussi que vous ayez des applications dont les fichiers .desktop n'aient pas été générés par E ou par menu_update : ceci arrive principalement pour les applications installées dans votre arborescence personnelle. Nous allons prendre l'exemple d'eclipse que nous avons installé dans : ~/bin/eclipse. Tout d'abord, lancez eclipse (ou tout autre application de votre choix) et éditez un fichier qui s'appellera eclipse.desktop dans ~/.e/e/applications/all.

```
01: [Desktop Entry]
02: Encoding=UTF-8
03: Name=Eclipse
04: X-Enlightenment-IconPath=/home/john/bin/Eclipse/eclipse/icon.xpm
05: Exec=/home/john/bin/eclipse #Chemin vers votre propre binaire.
06: Icon=eclipse.xpm #Icône associé à votre propre binaire.
07: Type=Application
```

3.4 Ajout de modules

Par défaut, même si vous n'avez pas compilé les modules cités plus haut, vous en avez quelques-uns de disponibles. Nous allons vous montrer la manipulation à effectuer pour activer ou désactiver un module.

Pour commencer, il y a un peu de terminologie à connaître :

- Les modules : ce sont des applications lancées dans E17 pour étendre ses fonctionnalités ;
- Le « *shelf* » : Le shelf est un conteneur de modules. Il est représenté par des barres le long des côtés de l'écran. L'agencement des shelves se fait via cette manipulation :
  - *Menu Contextuel de E* (clic gauche) -> *Configuration* -> *Shelves* (vous devez en avoir un par défaut) -> *Configure*
  - Dans ce panneau de configuration, vous pouvez agencer le shelf comme vous le souhaitez.

Avant de pouvoir ajouter des modules, il faut que vous configuriez le manager de modules afin de préciser si un module est actif ou non. Pour ce faire :

- *Menu Contextuel de E* (clic gauche) -> *Configuration* -> *Modules*

Vous vous trouvez dans un menu avec la liste des noms de modules sur la gauche. Pour en activer un ou en désactiver un, il vous suffit de choisir parmi une des deux cases à cocher : « Activé » ou « Désactivé »

ATTENTION

Il se peut que la traduction française ne soit pas complète.

Une fois cette manipulation faite, vous pouvez ajouter autant d'instances de modules à autant de shelfs que vous le souhaitez. Pour ce faire, faites la manipulation précédente concernant le shelf, et sur la droite, vous trouverez un bouton « Éditer le contenu ». Pressez ce bouton et ajoutez les modules que vous souhaitez.

Maintenant, il se peut que vous fassiez partie des utilisateurs de Gnome et que vous souhaitiez que les applications gtk que vous exécutez sous E17 utilisent votre style gtk favori. C'est le sujet de la prochaine partie.

3.5 Intégration de gtk

Pour obtenir une meilleure intégration de gtk au sein de votre nouvel environnement, il y a deux solutions :

- Si vous avez Gnome d'installé avec votre thème gtk préféré : décrivez un fichier desktop (cf. ci-dessus) permettant de lancer l'application gnome-settings-daemon dans le répertoire ~/.e/e/applications/all/, éditez le fichier .order du répertoire ~/.e/e/applications/startup afin d'y mettre cette entrée. Ce fichier desktop devra ressembler à ceci :

```
[Desktop Entry]
Name=gnome-settings-daemon
Application=gnome-settings-daemon
Exec=gnome-settings-daemon
Icon=""
```

- Relancez alors votre environnement et vous y verrez votre thème gtk de Gnome sous E.
- Vous pouvez par ailleurs utiliser le binaire :gtk-theme-switch2 (du paquet gtk-theme-switch sous Debian ou Ubuntu) disponible dans la majorité des distributions. Pour trouver des thèmes gtk sympas : **[15]**. Décompressez les dans ~/.themes et lancez le petit utilitaire installé précédemment. Nous vous laissons alors le choix de votre thème.

3.6 Thèmes

Tout d'abord, il se pourrait que le thème par défaut ne vous plaise pas du tout (un des points forts du Libre n'est-il pas la diversité ?), ou que vous souhaitiez en utiliser d'autres (ce qui est mon cas). Nous allons donc voir en détail comment faire pour en changer. Le site de la communauté propose des thèmes sympas **[10]** et **[11]**. Téléchargez les fichiers .edj et copiez-les dans le répertoire ~/.e/e/themes. Pour ensuite activer le thème :

- *Menu Contextuel de E* (clic gauche) -> *Configuration* -> *Thème*

A partir de là, il vous suffit de choisir le thème que vous voulez. Observons tout de même que si une partie du thème ne vous plaît pas, il vous est possible de l'éditer de cette manière :

- Faites : edje_decc montheme.edj ;
- Entrez dans le répertoire montheme et faites vos modifications. Ce répertoire est constitué d'images au format png et de fichiers source edje ;
- Une fois vos modifications terminées, exécutez : ./build.sh. Ce script vous regénère le fichier montheme.edj. Vous pouvez alors le déplacer dans le répertoire de thèmes de E pour commencer à l'utiliser.

NOTE

Vous pouvez passer par cette interface pour ajouter un thème via le bouton « Import ».

4. Petit exemple EFL

Dans tout cet article, nous vous avons parlé des Enlightenment Foundation Libraries. Il est temps que vous, lecteur intéressé par un peu de développement, y mettiez un peu du vôtre ;-). Nous allons exposer un simple tutoriel ci-dessous avec ewl dont le comportement sera des plus simples. L'interface est composée d'un champ de saisie et d'un bouton. Vous cliquez sur le bouton et une fenêtre modale apparaît par-dessus, avec comme texte ce que vous avez saisi précédemment.

Pour des raisons de lisibilité au sein du magazine, nous n'allons pas vous décrire les fichiers d'en-tête dans lesquels se trouvent seulement les déclarations des fonctions. Nous ne décrivons que les fichiers sources. Vous trouverez la totalité de l'exemple à cette adresse **[12]**.

4.1 Fichier principale.c

Ce fichier de sources se donne pour objectif de définir l'interface principale de cette petite application. Il y a un bouton de label « Saluer » et un champ de saisie de texte. Nous définissons, par ailleurs, une gestion d'évènements assez simple. Ceci est décrit par la définition de fonctions de *callback*.

```
01: /* Fonction appelée lors d'un clic sur le bouton "Saluer" */
02: void callback_ouverture_modale(Ewl_Widget *widget, void
*evenement, void *donnees)
03: {
04:     creer_popup(widget, evenement, donnees);
05: }
06:
07: /* Fonction appelée lors de la fermeture de la fenêtre
principale, donc de l'application */
08: void destroy_cb(Ewl_Widget *widget, void *evenement, void
*donnees)
09: {
10:     /* Destruction de la fenêtre */
11:     ewl_widget_destroy(widget);
12:     /* Arrêt de la boucle d'exécution */
13:     ewl_main_quit();
14: }
15:
16:
17: void creer_fenetre()
18: {
19:     Ewl_Widget *win, *vbox, *button, *saisie, *texte;
20:
21:     /* Création de la fenêtre principale */
22:     win = ewl_window_new();
23:     ewl_window_title_set(EWL_WINDOW(win), "Test EWL 1/2 :
saisie");
24:     ewl_window_name_set(EWL_WINDOW(win), "EWL_WINDOW");
25:     ewl_window_class_set(EWL_WINDOW(win), "EWLWindow");
26:     ewl_object_size_request(EWL_OBJECT(win), 300, 75);
27:     ewl_callback_append(win, EWL_CALLBACK_DELETE_WINDOW,
destroy_cb, NULL);
28:     ewl_widget_show(win);
29:
30:     /* Création d'une boîte d'alignement vertical pour
positionner nos widgets */
31:     vbox = ewl_vbox_new();
32:     ewl_container_child_append(EWL_CONTAINER(win), vbox);
33:     ewl_widget_show(vbox);
34:
35:     /* Création du label pour demander la saisie du prénom */
36:     texte = ewl_text_new();
37:     ewl_widget_name_set(texte, "texte0");
38:     ewl_text_selectable_set(EWL_TEXT(texte), TRUE);
39:     ewl_container_child_append(EWL_CONTAINER(vbox), texte);
40:     ewl_text_text_insert(EWL_TEXT(texte), "Entrez votre prénom
:", 0);
41:     ewl_widget_show(texte);
42:
43:     /* Création de la zone de saisie */
44:     saisie = ewl_entry_new();
45:     ewl_text_text_set(EWL_TEXT(saisie), " ");
46:     ewl_text_color_set(EWL_TEXT(saisie), 0, 0, 0, 255);
47:     ewl_object_padding_set(EWL_OBJECT(saisie), 5, 5, 0, 0);
48:     ewl_container_child_append(EWL_CONTAINER(vbox), saisie);
49:     ewl_widget_show(saisie);
50:
51:     /* Création du bouton d'appel de la popup */
52:     button = ewl_button_new();
53:     ewl_button_label_set(EWL_BUTTON(button), "Saluer");
54:     ewl_container_child_append(EWL_CONTAINER(vbox), button);
55:     ewl_callback_append(button, EWL_CALLBACK_CLICKED,
callback_ouverture_modale, saisie);
56:     ewl_widget_show(button);
57:
58:     /* Lancement de la boucle d'exécution */
59:     ewl_main();
60: }
```

4.2 Fichier popup.c

Ce fichier de sources représente l'interface de la fenêtre « popup ».

```
01: /* Fonction appelée à la fermeture de la popup */
02: void callback_fermeture(Ewl_Widget *widget, void *evenement,
void *donnees)
03: {
04:     Ewl_Embed *emb;
05:
06:     /* On récupere le widget de la fenêtre à fermer */
07:     emb = ewl_embed_widget_find(widget);
08:     /* Destruction de la fenêtre */
09:     ewl_widget_destroy(EWL_WIDGET(emb));
10: }
11:
12: /* Création de la popup */
13: void creer_popup(Ewl_Widget *widget, void *evenement, void
*donnees)
14: {
15:     Ewl_Embed *emb;
16:     Ewl_Widget *win, *vbox, *bouton, *texte;
17:
18:    /* On récupère un pointeur sur la fenêtre principale pour
définir la popup comme modale par rapport à elle */
19:     emb = ewl_embed_widget_find(widget);
20:
21:     /* Création de la popup */
22:     win = ewl_window_new();
23:     ewl_window_title_set(EWL_WINDOW(win), "Test EWL 2/2 :
salutation");
24:     ewl_object_size_request(EWL_OBJECT(win), 300, 50);
25:     ewl_callback_append(win, EWL_CALLBACK_DELETE_WINDOW,
callback_fermeture, NULL);
26:     ewl_widget_show(win);
27:
28:     /* Création d'une boîte d'alignement vertical pour
positionner nos widgets */
29:     vbox = ewl_vbox_new();
30:     ewl_container_child_append(EWL_CONTAINER(win), vbox);
```

```
31:     ewl_widget_show(vbox);
32:
33:     /* Création du label pour afficher la salutation */
34:     texte = ewl_text_new();
35:     ewl_widget_name_set(texte, "texte1");
36:     ewl_text_selectable_set(EWL_TEXT(texte), TRUE);
37:     ewl_container_child_append(EWL_CONTAINER(vbox), texte);
38:
39:     /* Affichage du texte de salutation + le prénom
        saisi-> portion de code pas sécurisé */
40:     int tailleChaine = 10 + strlen(
           ewl_text_text_get(EWL_TEXT(donnees)));
41:     char* chaine = (char*) malloc(sizeof(char)*tailleChaine);
42:     sprintf(chaine, "Bonjour %s!", ewl_text_text_get(
           EWL_TEXT(donnees)));
43:     chaine[strlen(chaine)] = '\0';
44:
45:     ewl_text_text_insert(EWL_TEXT(texte), chaine, 0);
46:     ewl_widget_show(texte);
47:
48:     /* Création du bouton de fermeture de la popup */
49:     bouton = ewl_button_new();
50:     ewl_button_label_set(EWL_BUTTON(bouton), "Fermer");
51:     ewl_callback_append(bouton, EWL_CALLBACK_CLICKED,
           callback_fermeture, NULL);
52:     ewl_container_child_append(EWL_CONTAINER(vbox), bouton);
53:     ewl_widget_show(bouton);
54:
55:     /* Mise de la popup en mode modal par rapport
        à la fenêtre principale */
56:     ewl_window_modal_for(EWL_WINDOW(win), EWL_WINDOW(emb));
57: }
```

Fichier main.c

C'est le point d'entrée de l'application.

```
01: int main(int argc, char* argv[])
02: {
03:     /* Initialisation de ewl */
04:     if (!ewl_init(&argc, argv))
05:     {
06:         fprintf(stderr, "Impossible
d'initialiser ewl\n");
07:         return 1;
08:     }
09:
10:     creer_fenetre();
11:
12:     return 0;
13: }
```

Conclusion

Ainsi se finit notre article, sur ce petit exemple écrit en ewl qui vous montre la simplicité d'utilisation des EFL. Vous avez maintenant entre les mains une version récente d'Enlightenment 17, n'hésitez pas en cas de problème à contacter la communauté via IRC **[14]** ou tout autre moyen de communication. A bientôt pour un voyage au cœur des bibliothèques de ce projet, à travers des exemples utilisant edje, evas, ecore, etk...

Jonathan Muller & Jean-Christophe Lauffer,

jonathan.muller@avisto.com, Consultant Avisto

lauffer.jc@gmail.com, Analyste Développeur

LIENS

- **[1]** Site du développeur principal, alias Rasterman : http://www.rasterman.com/
- **[2]** http://www.enlightenment.org/. Vous y trouverez les nouvelles du projet, ce qui est cassé dans les sources, les nouvelles fonctionnalités.
  - Get Enlightenment : http://www.get-e.org/
  - Sites officiels de la communauté française : http://www.enlightenment.org/ et http://fr.enlightenment.org/
- **[3]** Lecteur DVD en 17 lignes de C : http://www.supinfo-projects.com/fr/2005/presentation%5Fefl/1/
- **[4]** The Gimp : http://www.gimp-fr.org/news.php
- **[5]** Xine : http://xinehq.de/
- **[6]** Gstreamer : http://gstreamer.freedesktop.org/
- **[7]** Bump Mapping : http://en.wikipedia.org/wiki/Bump_mapping
- **[8]** Vidéo de chargement de E : http://www.rasterman.com/files/e17_movie-00.avi
- **[9]** Document de norme du format .desktop : http://www.freedesktop.org/wiki/Standards_2fdesktop_2dentry_2dspec
- **[10]** Page générale de thèmes : http://www1.get-e.org/Themes/E17/
- **[11]** Thème Winter : http://www.rephorm.com/news/tag/winter
- **[12]** Exemple fourni dans cet article : http://www.drylm.org/e17/examples/ewl1.tar.gz
- **[13]** Interface de programmation : http://fr.wikipedia.org/wiki/Interface_de_programmation
- **[14]** Channels IRC : #e.fr@irc.freenode.net et #e@irc.freenode.net
- **[15]** Thèmes Gtk : http://www.gnome-look.org/

Une nouvelle disposition de clavier français pour Xorg

Camille Huot

EN DEUX MOTS **Xorg contient maintenant une nouvelle disposition de touches pour les claviers français AZERTY. L'utilisation de cette disposition permet l'accès rapide à de nombreux symboles supplémentaires qui ne sont pas dessinés sur les touches de votre clavier. La nouvelle version supporte enfin des caractères Unicode qui n'étaient pas accessibles jusqu'à présent.**

Qu'est-ce qu'une disposition de touches Xorg ?

C'est un ensemble de quelques fichiers textes situés dans l'arborescence en dessous du répertoire `/usr/share/keymaps/` qui permettent à Xorg d'associer chaque touche de votre clavier à un symbole Xorg. Ce symbole peut-être une lettre, comme « a », « b », « c »... ou un symbole spécial, comme « XF86AudioRaiseVolume », généralement associé à la touche de votre clavier multimédia qui permet d'augmenter le volume sonore. Par exemple, il est responsable du fait que lorsque vous maintenez enfoncées les touches Majuscule et « a », vous obtenez un « A ». Chaque type de clavier différent doit avoir la bonne disposition de clavier Xorg correctement installée, sinon vous ne pourrez pas utiliser votre clavier dans de bonnes conditions.

Un peu d'histoire

En France, nous utilisons généralement des claviers de type « AZERTY », par référence aux premières lettres du clavier. En fait, cette désignation n'est pas suffisante, car les Belges, par exemple, utilisent également des claviers AZERTY qui ne sont pas pour autant complètement compatibles avec nos claviers français.

La disposition de touches des claviers français que l'on trouve sous Xorg, qui est bien sûr héritée de XFree86, appelée « fr-latin1 », apportait déjà de nombreuses améliorations par rapport à la disposition que l'on trouve toujours actuellement sous Windows. La majorité des touches lettrées produisent ainsi un caractère spécial lorsqu'on l'utilise en maintenant la touche [AltGr] enfoncée. Par exemple, [AltGr]+[a] produit le caractère « æ », [AltGr]+[z] produit « « »... Ce genre de combinaison de touches facilite grandement la saisie de ces caractères et permet de répandre plus rapidement leur utilisation, ce qui produit des textes plus conformes à la typographie française. En effet, ce n'est pas correct d'utiliser les guillemets américains « " » dans un texte français, ni d'écrire œuf « oeuf » ou encore de ne pas accentuer les majuscules. Cela passait à l'époque de la machine à écrire parce qu'on n'avait pas le choix, mais cette époque est révolue depuis longtemps. Nous avons tous les outils nécessaires, il suffit de s'y mettre. Justement, cet article va nous aider dans ce sens.

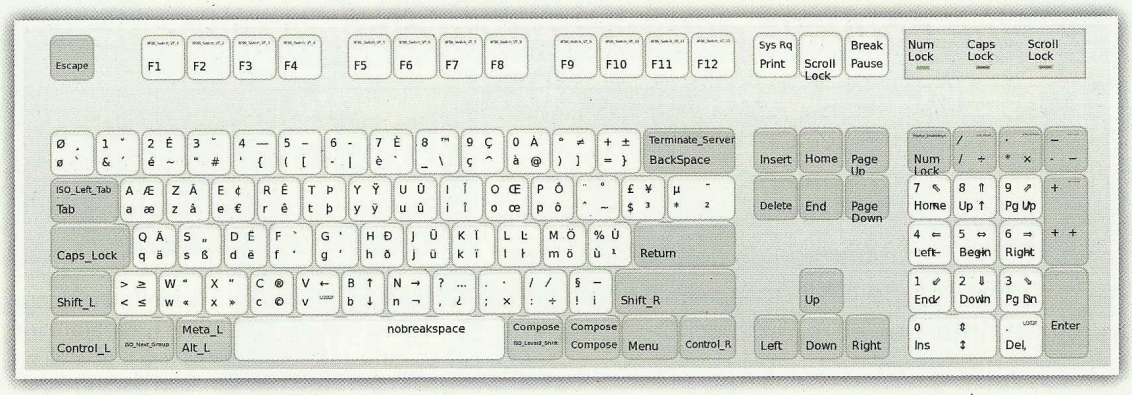
Le jeu de caractères (*charset* en anglais) Latin1, aussi appelé « ISO-8859-1 », est limité à 256 caractères (contrainte technique) et lorsque l'euro devient la monnaie européenne, il faut rajouter ce symbole dans la liste des caractères. L'ISO sort alors l'ISO-8859-15 (Latin9), une légère modification du ISO-8859-1, en ajoutant notamment les caractères « € » et « œ ».

C'est à ce moment qu'est créée la disposition de clavier fr-latin9, qui est une refonte totale des touches accessibles par [AltGr], en tenant compte des changements apportés par le nouveau jeu de caractères Latin9. La disposition fr-latin9 est, en principe, celle que nous utilisons en ce moment. Les caractères spéciaux ont été placés plus intuitivement, les voyelles accentuées (circonflexe, tréma...) sont accessibles directement par [AltGr] en plus de la méthode habituelle par les touches mortes ([AltGr]+[i] donne « î », [AltGr]+[Maj]+[i] donne « ï »), les majuscules accentuées comme le « É » apparaissent ([AltGr]+[Maj]+[é]), ainsi qu'une touche morte accent aigu ([AltGr]+[&]+[a] donne « á »), la touche [²] devient « œ »... bref, pour peu qu'on utilise régulièrement ces caractères pour faire l'effort d'apprendre ces combinaisons par cœur, c'est un vrai gain de temps lors de la frappe.

Présentation de la nouvelle disposition

Durant le mois de septembre, le mainteneur de la disposition fr-latin9 a lancé un appel à la communauté Linux et Xorg pour la faire évoluer. Concrètement, le but était de corriger les éventuels problèmes de la version actuelle, de demander des suggestions aux utilisateurs, d'intégrer des caractères Unicode (puisque l'UTF-8 est le jeu de caractères par défaut dans de nombreuses distributions maintenant et qu'il permet d'utiliser un bien plus grand nombre de caractères, c'est-à-dire la quasi-totalité des alphabets du monde) et de rajouter des touches sur le pavé numérique. L'appel fut entendu et après quelques semaines de discussions, un consensus a été atteint et les responsables Freedesktop (Xorg) ont accepté la nouvelle carte.
Nommée « fr-oss », pour *Open Source Software* et pour ne plus lier le nom de la disposition à un jeu de caractères, la nouvelle carte pour les claviers français sous Xorg a pas mal évolué depuis fr-latin9.

Contrairement à ce qu'on pourrait croire, le clavier AZERTY n'est pas conçu pour la frappe rapide. Cette disposition remonte au XIXe siècle, et est dérivée du QWERTY anglais, spécifiquement étudié pour éviter les risques de blocage des machines à écrire mécaniques.



À la demande générale, le symbole « æ » est revenu sur la touche [A] comme en fr-latin1, et le « œ » est passé sur la touche [O]. Il suffit de rajouter la touche [Maj] à la combinaison produite pour obtenir la majuscule du caractère. Les voyelles avec accent circonflexe se situent soit sur la touche elle-même, soit sur la touche immédiatement à droite et les voyelles avec tréma sont placées généralement en dessous de la voyelle. Parmi les nouveaux symboles, on note particulièrement le « ™ » sur la touche [8], le « ≠ » sur la touche [°], les « ≤ » et « ≥ » sur la touche [<], les différents tirets « — », « – » et « - » sur les touches [4], [5] et [6] du pavé principal… (En parlant de trois petits points, le symbole « … » est rajouté sur la touche [?].) Le symbole « € » est définitivement installé sur la touche [e], avec son « ¢ » pour les centimes, le yen (¥) rejoint le dollar et la livre sterling, les guillemets français et anglais se stabilisent sur les touches [W] et [X].

Une grosse nouveauté est l'utilisation de [Maj] et [AltGr] avec le pavé numérique. On note l'insertion de tous les symboles mathématiques courants sous plusieurs formes selon la combinaison de touches effectuée. Prenons la touche [*], qui représente une multiplication en informatique. Avec [AltGr], le caractère affiché est [×], qui ressemble nettement plus à un signe « multiplier » que l'astérisque ! Avec [Maj], le caractère est « · », ce petit point qu'on utilise pour représenter les produits dès le lycée. Sur les touches chiffrées, on pourra produire des flèches pointant vers toutes les directions, simples ou doubles corps. Une amélioration que les comptables du libre apprécieront certainement : [AltGr]+[Suppr] du pavé numérique produit une virgule, très pratique pour écrire des nombres à virgule à la française, même en dehors d'Openoffice qui fait déjà ça automatiquement.

Installer la disposition fr-oss

Cette nouvelle carte pour les claviers français a été intégrée dans le CVS de Freedesktop fin septembre et fera donc partie de la prochaine version de Xorg. À l'heure où vous lisez ces lignes, la version 0.9 de `xkeyboard-config` devrait être sortie et devrait contenir la nouvelle carte. Vous n'avez qu'à installer au minimum cette version et passer directement à la section suivante.

Si ce n'est pas encore le cas, voici comment vous pouvez installer les modifications sur la version 0.8 de `xkeyboard-config` :

Téléchargement de l'archive et des patchs :

```
$ wget http://xlibs.freedesktop.org/xkbdesc/xkeyboard-config-0.8.tar.bz2
$ wget --no-check-certificate -O keypad.patch \
https://bugs.freedesktop.org/attachment.cgi?id=7053
$ wget --no-check-certificate -O translation.patch \
https://bugs.freedesktop.org/attachment.cgi?id=7054
$ wget --no-check-certificate -O keyboard.patch \
https://bugs.freedesktop.org/attachment.cgi?id=7083
```

Décompression et patch

```
$ tar xvjf xkeyboard-config-0.8.tar.bz2
$ cd xkeyboard-config-0.8
$ mv symbols/fr symbols/fr.orig
$ iconv -f iso-8859-15 -t utf8 symbols/fr.orig > symbols/fr
$ patch -p1 < ../keypad.patch
$ patch -p1 < ../keyboard.patch
$ patch -p1 < ../translation.patch
```

Compilation et installation

```
$ ./configure --with-xkb-base=/usr/share/X11/xkb
$ make
$ sudo make install
```

Activer la disposition fr-oss

Pour activer cette nouvelle disposition de clavier dans Xorg pour tous les utilisateurs de la machine (au niveau global), rien de plus simple. Il suffit d'ouvrir le fichier `/etc/X11/xorg.conf` en tant que *root* et de faire en sorte que la section servant à configurer le clavier ressemble à ceci :

```
Section "InputDevice"
        Identifier  "Keyboard0" # permet d'identifier le périphérique
        Driver      "kbd"
# [...]
        Option      "XkbModel" "pc105"
        Option      "XkbLayout" "fr"
        Option      "XkbVariant" "oss"
EndSection
```

Ensuite, vous n'avez qu'à redémarrer le serveur X pour vérifier que les changements ont bien été pris en compte. Concrètement, il suffit de vous déconnecter de votre environnement graphique et votre gestionnaire de session démarrera un nouveau serveur X pour que vous vous reconnectiez.

Si vous n'êtes pas root de votre machine, si vous ne voulez activer fr-oss que pour un seul utilisateur ou bien si vous voulez simplement tester cette nouvelle carte avant de l'activer pour de bon, vous pouvez charger dynamiquement la disposition de touches en tapant la commande :

```
setxkbmap fr oss
```

Pour tester la nouvelle configuration de votre clavier, faites [AltGr]+[z], cela devrait afficher un « â ».

Si cela ne vous plaît pas, vous pouvez revenir à tout moment à la disposition fr-latin9 (en supposant que vous utilisez celle-ci) :

```
setxkbmap fr latin9
```

Modifier des touches via Xmodmap

Si vous n'êtes pas pleinement satisfait de votre disposition, si vous voulez intervertir une touche ou deux, ou placer des caractères que vous utilisez souvent, vous pouvez utiliser Xmodmap pour modifier la carte du clavier. Il suffit de placer les directives voulues dans le fichiers `~/.Xmodmap` et celles-ci seront chargées automatiquement lors du chargement de X par Xinit (dans le cas où vous démarrez votre session avec Startx). Si vous utilisez un gestionnaire de sessions graphique (KDM, GDM, XDM…), peut-être que cela ne marchera pas automatiquement. Dans ce cas, il faut rajouter la ligne suivante dans votre fichier de démarrage `~/.xsession` :

```
xmodmap ~/.Xmodmap
```

Voici un exemple de fichier `.Xmodmap` qui a servi avec la disposition fr-latin9 pour intervertir les symboles « euro » et « cent », placés malencontreusement sur la touche [E], avec les symboles « ê » et « ë », placés comme par hasard sur la touche [$]. Autant remettre l'« € » et le « ¢ » avec leurs camarades « $ » et « £ », non ?

```
keycode  26 = e E ecircumflex ediaeresis ecircumflex ediaeresis
keycode  35 = dollar sterling EuroSign cent EuroSign cent
```

Les codes 26 et 35 sont les codes des touches [E] et [$] respectivement. Vous pouvez obtenir le code de n'importe quelle touche, y compris de la souris, en utilisant Xev, l'intercepteur d'événement de Xorg. Lancez Xev, mettez-vous dans la petite fenêtre qu'il vient d'ouvrir et tapez une touche. Tout un tas d'informations à propos de cette touche apparaîtront : lors de l'appui sur la touche, puis lors de la libération de celle-ci. Ce sont deux événements distincts pour Xorg. Par exemple, si je tape la touche A :

Lors de l'appui :

```
KeyPress event, serial 31, synthetic NO, window 0x4600001,
    root 0x44, subw 0x0, time 4199053440, (92,79),
root:(1095,484),
    state 0x10, keycode 24 (keysym 0x61, a), same_screen YES,
    XLookupString gives 1 bytes: (61) "a"
    XmbLookupString gives 1 bytes: (61) "a"
    XFilterEvent returns: False
```

Lors de la libération :

```
KeyRelease event, serial 31, synthetic NO, window 0x4600001,
    root 0x44, subw 0x0, time 4199053473, (92,79),
root:(1095,484),
    state 0x10, keycode 24 (keysym 0x61, a), same_screen YES,
    XLookupString gives 1 bytes: (61) "a"
```

On note bien que le code de la touche [A] est 24 : keycode 24.

Chaque ligne permet donc d'assigner une série de caractères à une touche. Dans l'ordre :

```
keycode <code de la touche> = <touche> <touche+Maj> <touche+Mode_
Switch> <touche+Maj+Mode_Switch> <touche+AltGr> <touche+AltGr+Maj>
```

La touche `Mode_Switch` n'est pas présente sur nos claviers. Elle représente le [Alt] de droite. Nous, nous avons un [AltGr] à la place.

Camille Huot,

camille@huot.name

RÉFÉRENCES

- ISO-8859-1 (Latin1) : http://en.wikipedia.org/wiki/ISO/IEC_8859-1
- ISO-8859-15 (Latin9) : http://en.wikipedia.org/wiki/ISO/IEC_8859-15
- Appel à évolution de la disposition fr-latin9 : http://linuxfr.org/2006/09/13/21322.html

La fameuse erreur de frappe

Avez-vous déjà vu ce genre d'erreur :

```
$ ls -Rl / | more
bash:  more: command not found
```

C'est une faute de frappe qui arrive fréquemment lorsqu'on utilise la disposition fr-latin9. Lorsque vous tapez l'espace qui suit le *pipe* « | », votre pouce peut traîner sur la touche [AltGr]. Ceci peut vous amener à produire la combinaison de touches [AltGr]+[Espace], ce qui génère une espace insécable, qui est un caractère différent de l'espace normale. Bash ne le reconnaît donc pas, le prend comme un caractère ordinaire et croit que vous voulez lancer le programme « more » (notez l'espace devant), programme qui n'existe pas, en principe. D'où l'erreur.

Ce comportement a été remonté par de nombreux utilisateurs et a donc été supprimé dans la nouvelle version. Cependant, il est toujours possible de produire une espace insécable, pour cela il faut faire [AltGr]+[Maj]+[Espace], combinaison qui est beaucoup plus difficile à faire par mégarde !

Et sous Windows ?

Si vous passez régulièrement de Linux à Windows, par exemple si vous utilisez encore Windows au travail, vous voudrez peut-être quand même profiter de ces évolutions fantastiques qui vous font gagner du temps lors de la frappe. Eh bien, ces dispositions ont été portées sous Windows grâce au *Keyboard Layout Creator* de Microsoft.

Vous pouvez télécharger ces dispositions sur ce site : http://www.cameuh.net/projects/fr-latin9/

Il suffit d'installer le fichier `.msi` fourni et ensuite d'aller dans les préférences régionales pour changer la langue du clavier. La nouvelle disposition figurera dans la liste.

Vous pouvez également télécharger le source de la disposition, disponible sur le même site, et procéder à vos propres modifications en utilisant l'outil de Microsoft (disponible en téléchargement ici : http://www.microsoft.com/globaldev/tools/msklc.mspx).

Yafray, le moteur de rendu photoréaliste libre
Une première approche

Olivier Saraja

EN DEUX MOTS **Yafray est un moteur de rendu libre dont l'objectif est le photoréalisme. Bien que somme toute classique dans ses fonctionnalités, bien utilisé, il s'avère rapide et puissant, capable de résultats de très bonne tenue. Il en devient le compagnon de route de divers modeleurs, à l'image de Wings3D ou Blender lui-même.**

A l'instar de Povray, Yafray repose sur un langage de description de scène (*Scene Description Language* ou SDL) pour l'instruire de ce qu'il faut rendre. Le fichier de description de scène est tout simplement un fichier XML formaté de façon spécifique. L'objectif de cette mini-série d'articles sur Yafray est d'apprendre à maîtriser son langage de description et l'utiliser de façon pertinente, en approfondissant notre connaissance de ses paramètres et rouages internes.

Yafray : le lancer de rayon pour les masses

1. Compilation et installation de Yafray

L'installation de Yafray devrait être triviale si vous utilisez les paquets pré-compilés proposés sur la page Downloads du projet (http://www.yafray.org/index.php?s=2). Malheureusement, ils ne fonctionnent pas à tous les coups ; par exemple, les paquets pré-compilés pour Debian Etch ne fonctionnent pas sur une Ubuntu. De plus, en tant que moteur de rendu, pour bénéficier des temps de calcul les plus courts possibles, optimiser le moteur de rendu en fonction de votre classe de processeur est une bonne idée. Compiler et installer soi-même Yafray est donc fortement conseillé, et ne devrait pas poser de souci majeur au lecteur « moyen » de *GNU/Linux Magazine*.

1.1 Outils et dépendances

Yafray est soumis à plusieurs dépendances qui nécessiteront donc l'installation de bibliothèques et outils variés pour permettre sa compilation. Les outils nécessaires comprennent bien évidemment le compilateur c++ `gcc`, ainsi que les scripts `scons`. L'installation de `g++` est également absolument nécessaire, et est souvent oubliée par les apprentis linuxiens se lançant dans la compilation de Yafray sans trop lire la documentation.

Les dépendances devant être résolues comprennent `libc`, `libgcc` et `libstdc++`, qui sont des dépendances courantes de gcc et g++. Leur installation risque donc d'être transparente pour vous. En revanche, vous devrez explicitement veiller à l'installation de `libjpeg`, `libopenexr` et `zlib`.

1.2 Compilation

Vous téléchargerez les sources de Yafray sur la page Downloads déjà citée du projet. Après avoir décompacté l'archive .tar.gz sur votre disque, vous vous rendrez dans le répertoire `yafray` ainsi créé et taperez :

```
$ scons arch=athlon-xp
```

pour lancer la compilation. Bien sûr, vous adapterez la valeur de `arch` en fonction du processeur de votre machine. Les valeurs les plus courantes devraient être `i686`, `pentium4` ou `athlon-xp`. La liste complète des possibilités se trouve ici **http://gcc.gnu.org/onlinedocs/gcc-3.4.3/gcc/i386-and-x86_002d64-Options.html#i386-and-x86_002d64-Options**. Si votre processeur ne se trouve vraiment pas dans la liste, lancez la compilation grâce à la commande `scons` seule.

1.3 Installation des fichiers

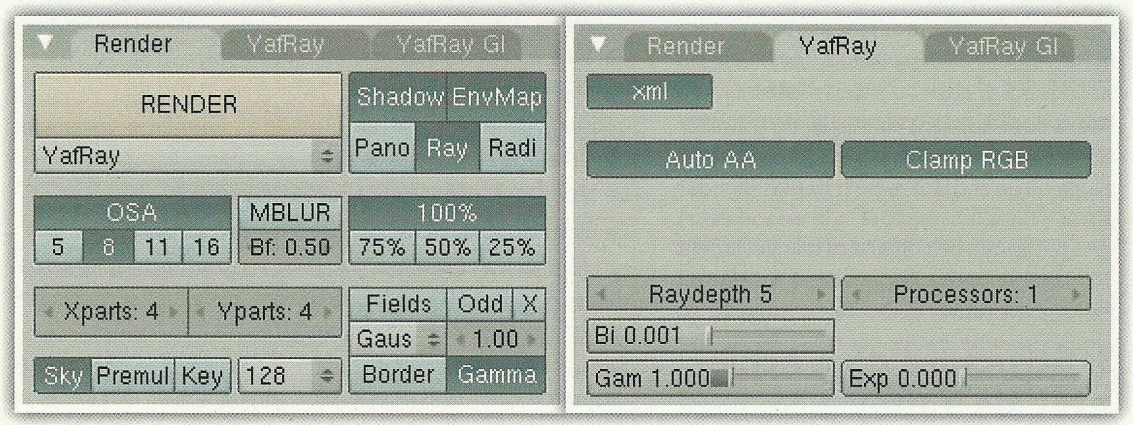
Pour l'étape suivante, vous aurez besoin des privilèges du super-utilisateur. Il vous faudra utiliser la commande suivante :

```
$ scons arch=athlon-xp install
```

Pensez à bien spécifier la même architecture que précédemment, sinon `scons` recompilera à nouveau tous les fichiers, mais sans optimisation particulière, avant de les installer. Le répertoire d'installation par défaut est `/usr/local`. Les exécutables de Yafray seront donc localisés dans `/usr/local/bin` tandis que les librairies seront dans `/usr/local/lib`. Pour parfaire et conclure l'installation de Yafray, il vous faudra ajouter `/usr/local/lib/` au fichier `/etc/ld.so.conf` et utiliser la commande `ldconfig` avec les privilèges du super-utilisateur.

2. Usage de Yafray à partir de Blender (et création de votre premier fichier XML)

Ce n'est pas l'objet de ces articles, mais quelques mots sur l'usage conjoint de Blender et Yafray peuvent vous aider dans votre apprentissage de ce dernier. Ouvrez une session de Blender. Allez dans le menu *Scene* (touche [F10]) et dans le panneau *Render*, sélectionnez *YafRay* à la place de *Blender Internal*. Deux nouveaux onglets se greffent au panneau *Render*.



Les principaux onglets relatifs au rendu avec Yafray depuis Blender

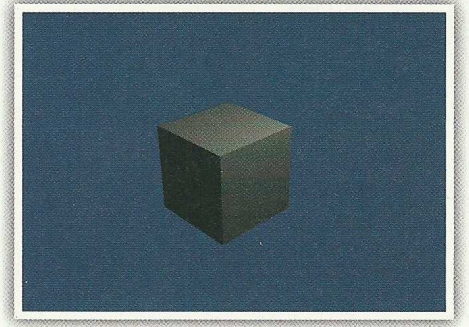
Jetez immédiatement un œil à l'onglet *YafRay*, et vérifiez que le bouton *xml* est activé. C'est grâce à lui qu'au moment du rendu seront créés les fichiers vous permettant de paramétrer finement Yafray. Faîtes le test avec la scène par défaut de Blender, en retournant dans l'onglet *Render* et en cliquant sur le bouton éponyme. La console de Blender nous renseigne sur les actions alors entreprises :

```
Starting scene conversion.
Scene conversion done.
No export directory set in user defaults!
Will try TEMP instead: /tmp
Yafray found at : /usr/local/bin/
COMMAND: /usr/local/bin/yafray -c 2 "/tmp/YBtest.xml"
Starting YafRay ...
Loading grammar ...
Starting parser ...
Parsing OK
Loading plugins from '/usr/local/lib/yafray'...
Registered arealight
Registered basicblocks
Registered basicshaders
Registered blendershaders
Registered globalphotonlight
Registered HDRI background
Registered hemilight
Registered mix block
Registered pathlight
Registered photonlight
Registered pointlight
Registered Shader Background
Registered softlight
Registered spherelight
Registered spotlight
Registered sss
Registered sunlight
Registered sunsky
found 18 plugins!
[Loader]: Added shader MAMaterial
starting build of kd-tree
=== kd-tree stats (0s) ===
primitives in tree: 12
interior nodes: 1 / leaf nodes: 2 (empty: 1 = 50%)
[Loader]: Added object OBCube
[Loader]: Added pointlight light LAMP1
[Loader]: Added constant background world_background
[Loader]: Added camera MAINCAM
Using a world resolution of 0.00114286 per unit
Rendering with 5 raydepth
2 anti-alias passes and 4 minimum samples per pass, 8 samples
total.
Building bounding tree ... Object count= 1
OK
Light setup ...
Setting up lights ...
Finished setting up lights

Launching 2 threads

Render pass: [##############]
Saving Targa file as /tmp/YBtest.tga
OK
Yafray completed successfully
```

Vous noterez en particulier que le répertoire par défaut de Yafray est `/tmp/`, et que l'image de sortie est automatiquement enregistrée au format Targa sous le nom `Ybtest.tga`. La scène Blender, convertie au langage de description de scène propre à Yafray, a été enregistrée sous forme de fichier `.xml` dans ce même répertoire.



Le cube par défaut de Blender, rendu par Yafray : le début de la passion !

Ce fichier peut être ouvert et modifié à l'aide de n'importe quel éditeur de texte. Ce qui est particulièrement intéressant, c'est qu'il est parfaitement structuré, et donc relativement facile à parcourir. Nous insisterons sur le « relativement ». En effet, chaque maillage de la scène est décrit par une succession de points (décrivant les sommets du maillage) et de faces (décrivant les facettes du maillage). Pour les objets les plus simples, les fichiers `.xml` résultants

resteront de taille modeste (quelques kilo-octets) ; pour d'autres scènes, il sera courant d'atteindre plusieurs méga-octets !

Il sera donc souvent plus facile de travailler sur la base d'un fichier `.xml` généré par un *plugin* ou un script d'export depuis Blender et/ou Wings3D (ou tout autre modeleur) vers Yafray, que de construire manuellement un fichier propre à Yafray, bien que cela soit bien sûr possible.

3. Structure d'un fichier XML de Yafray

Si vous jetez un œil sur le fichier `.xml` correspondant à la scène par défaut, vous risquez d'être impressionné par la quantité d'informations qui y figure. Cette sensation légitime ne durera qu'un court instant, car le fichier est clairement hiérarchisé et finalement facile à parcourir. En effet, la scène est décrite grâce à un système de balises qui ne sont pas sans rappeler celles du langage HTML ; il y a donc des balises ouvrantes et des balises fermantes, et entre celles-ci, des paramètres. Les balises et leur contenu forment un bloc, et chaque bloc instruit le moteur de rendu sur un élément de la scène. Le squelette d'une scène Yafray peut donc ressembler à quelque chose comme cela :

```
<scene>
    <shader>
    </shader>
    <transform>
        <object>
        </object>
    </transform>
    <light>
    </light>
    <camera>
    </camera>
    <filter>
    </filter>
    <background>
    </background>
    <render>
    </render>
</scene>
```

Le bloc `scene` est celui qui englobe tous les autres. Il permet de délimiter clairement la scène qui doit être rendue.

Le bloc `shader` définit un matériau. Il peut bien évidemment y en avoir plusieurs (au moins un par objet présent dans la scène), mais chaque bloc `shader` doit être défini avant qu'il n'y soit fait appel, généralement par un bloc `object`. Généralement, un bloc `shader` contient un bloc `attributes`.

Le bloc `transform` permet de définir la matrice de transformation de l'objet (localisation, rotations, échelle...). L'objet est ensuite défini dans un bloc qui lui est propre, mais contenu par le bloc `transform`.

Le bloc `object` contient la géométrie de l'objet rendu. Il contient généralement un bloc `mesh`, lui-même contenant un bloc `points` et un bloc `faces`.

Le bloc `light` définit une source de lumière.

Le bloc `background` définit l'arrière-plan du rendu.

Le bloc `camera` définit non seulement la position et la direction d'observation de la caméra de la scène, mais aussi la résolution de l'image rendue ou les paramètres de flou focal.

Le bloc `render` permet de définir les paramètres de rendu, la qualité de celui-ci ainsi que, bien sûr, un fichier de sortie.

La notion la plus importante à retenir est que le moteur de rendu va parcourir ce fichier du haut vers le bas, et que l'ordre d'occurrence des éléments a son importance. En effet, si un bloc fait appel à des éléments définis dans un autre bloc, ce bloc doit impérativement être défini avant, c'est-à-dire, se trouver en amont dans le fichier. C'est le plus souvent le cas avec les objets, qui font appel à un *shader*, celui-ci devant être défini avant que le moteur de rendu n'arrive au bloc propre de l'objet. C'est également le cas du bloc `render`, qui peut faire appel à un arrière-plan ; le bloc `background` correspondant devra donc être défini en amont du bloc `render`.

4. Quelques bases rapides et utiles

Plutôt que d'attaquer, bille en tête, la syntaxe particulière de Yafray, nous allons survoler quelques-uns des éléments les plus couramment utilisés. Ainsi, vous appréhenderez plus facilement les prochaines parties de cette mini-série. Pour cela, il va nous falloir un exemple concret, et quoi de mieux que le fichier XML de la scène par défaut de Blender ? Si vous avez suivi scrupuleusement les instructions de section 2 de cet article, vous devriez trouver un fichier nommé `Ybtest.xml` dans votre répertoire `/tmp/`. Ouvrez-le avec votre éditeur de texte préféré, et découvrez-en le contenu.

```
<scene>
<shader type="blendershader" name="MAMaterial" >
        <attributes>
            <color r="0.800000" g="0.800000" b="0.800000" />
            <specular_color r="1.000000" g="1.000000" b="1.000000" />
            <mirror_color r="1.000000" g="1.000000" b="1.000000" />
            <diffuse_reflect value="0.800000" />
            <specular_amount value="0.500000" />
            <alpha value="1.000000" />
            <emit value="0.000000" />
            <matmodes value="traceable shadow" />
            <diffuse_brdf value="lambert" />
            <specular_brdf value="blender_cooktorr" />
            <hard value="50" />
        </attributes>
</shader>
<transform m00="1.000000" m01="0.000000" m02="0.000000" m03="0.000000"
         m10="0.000000" m11="1.000000" m12="0.000000" m13="0.000000"
         m20="0.000000" m21="0.000000" m22="1.000000" m23="0.000000"
         m30="0.000000" m31="0.000000" m32="0.000000" m33="1.000000">
<object name="OBCube" shadow="on"  shader_name="MAMaterial" >
        <attributes>
        </attributes>
        <mesh autosmooth="0.1" has_orco="on" >
                <points>
                <p x="1.000000" y="1.000000" z="-1.000000" />
```

```
                    <p x="0.999999" y="1.000000" z="-1.000000" />
                    <p x="1.000000" y="-1.000000" z="-1.000000" />
                    <p x="0.999999" y="-0.999999" z="-1.000000" />
                    <p x="-1.000000" y="-1.000000" z="-1.000000" />
                    <p x="-1.000000" y="-0.999999" z="-1.000000" />
                    <p x="-0.999999" y="1.000000" z="-1.000000" />
                    <p x="-0.999999" y="1.000000" z="-1.000000" />
                    <p x="1.000000" y="1.000000" z="0.999999" />
                    <p x="1.000000" y="0.999999" z="1.000000" />
                    <p x="-1.000000" y="1.000000" z="1.000000" />
                    <p x="-1.000000" y="1.000000" z="1.000000" />
                    <p x="-1.000000" y="-1.000000" z="1.000000" />
                    <p x="-1.000000" y="-0.999999" z="1.000000" />
                    <p x="0.999999" y="-1.000000" z="1.000000" />
                    <p x="0.999999" y="-1.000000" z="1.000000" />
                    </points>
          <faces>
                    <f a="0" b="2" c="4" shader_name="MAMaterial" />
                    <f a="4" b="6" c="0" shader_name="MAMaterial" />
                    <f a="8" b="10" c="12" shader_name="MAMaterial" />
                    <f a="12" b="14" c="8" shader_name="MAMaterial" />
                    <f a="0" b="8" c="14" shader_name="MAMaterial" />
                    <f a="14" b="2" c="0" shader_name="MAMaterial" />
                    <f a="2" b="14" c="12" shader_name="MAMaterial" />
                    <f a="12" b="4" c="2" shader_name="MAMaterial" />
                    <f a="4" b="12" c="10" shader_name="MAMaterial" />
                    <f a="10" b="6" c="4" shader_name="MAMaterial" />
                    <f a="8" b="0" c="6" shader_name="MAMaterial" />
                    <f a="6" b="10" c="8" shader_name="MAMaterial" />
                    </faces>
          </mesh>
</object>
</transform>
<light type="pointlight" glow_intensity="0.000000" glow_
offset="0.000000" glow_type="0" name="LAMP1" power="29.999983"
cast_shadows="on" >
          <from x="4.076245" y="1.005454" z="5.903862" />
          <color r="1.000000" g="1.000000" b="1.000000" />
</light>
<background type="constant" name="world_background" >
          <color r="0.056563" g="0.220815" b="0.400000" />
</background>
<camera name="MAINCAM" type="perspective" resx="800" resy="600"
          focal="1.093750" aspect_ratio="1.000000"
          dof_distance="0.000000" aperture="0.000000" use_qmc="on"
          bokeh_type="disk1" bokeh_bias="uniform" bokeh_
          rotation="0.000000" >
          <from x="7.481132" y="-6.507640" z="5.343665" />
          <to x="6.826270" y="-5.896974" z="4.898420" />
          <up x="7.163761" y="-6.195171" z="6.239008" />
</camera>
<render camera_name="MAINCAM"
          raydepth="5" gamma="1.000000" exposure="0.000000"
          AA_passes="2" AA_minsamples="4"
          AA_pixelwidth="1.5" AA_threshold="0.05" bias="0.001000"
          clamp_rgb="on"
          background_name="world_background"
 >
          <outfile value="/tmp/YBtest.tga" />
</render>
</scene>
```

Un bloc donné peut contenir différents paramètres, chacun compris entre un < ouvrant et un /> fermant. Par exemple, à l'intérieur du bloc `camera` se trouve le paramètre `from` suivant :

```
<from x="7.481132" y="-6.507640" z="5.343665" />
```

Le paramètre peut posséder plusieurs attributs (par exemple, ici : `x=" "`, `y=" "` et `z=" "`), chacun suivi d'un signe égal = et d'une valeur entre guillemets ". La balise du bloc peut elle-même comprendre divers attributs. Par exemple, le bloc suivant :

```
<camera name="MAINCAM" type="perspective" resx="800" resy="600" />
```

permet de définir une caméra unique, en lui donnant un nom (`name=" "`), un type de rendu (`type=" "`) et une résolution de rendu (`resx=" "` et `resy=" "`). Il est en effet possible de définir plusieurs caméras (au travers de plusieurs blocs), chacune avec des attributs qui lui seront particuliers.

4.1 Le bloc transform

Il s'agit d'une matrice 4x4 de transformation « classique ». La portion 3x3 en haut à gauche correspond à la matrice de transformation (rotation, échelle, cisaillement) et le vecteur colonne (`m03`, `m13`, `m23`) représente la translation. La dernière ligne est « inutilisée » et reste toujours (`0`, `0`, `0`, `1`).

La diagonale (`m00`, `m11`, `m22`) représente l'échelle dans les directions (respectivement) `x`, `y` et `z`.

Construire une matrice de transformation « à la main » est très difficile, car la simple rotation autour d'un axe va affecter plusieurs éléments de la matrice, chacun de ceux-ci étant une combinaison des sinus ou cosinus des angles de rotation.

Par exemple, une simple rotation de 45° autour de l'axe z donne le bloc `<transform>` suivant, avec les autres rotations, localisation et échelle à leurs valeurs de base :

```
<transform m00="0.707107" m01="-0.707107" m02="0.000000" m03="0.000000"
           m10="0.707107" m11="0.707107" m12="0.000000" m13="0.000000"
           m20="0.000000" m21="0.000000" m22="1.000000" m23="0.000000"
           m30="0.000000" m31="0.000000" m32="0.000000" m33="1.000000">
```

En conséquence, la manipulation directe du bloc `<transform>` sera plutôt réservée à des mathématiciens enthousiastes ou avertis ; l'utilisateur moyen s'en remettra plutôt à l'exporteur XML (depuis l'application Blender ou Wings3d de son choix) pour la détermination de ces valeurs.

4.2 Le bloc camera

La syntaxe de ce bloc s'approche plus de ce que l'on peut voir dans des langages de description de scènes comme celui de Povray. Nous noterons en particulier les paramètres `resx=" "` et `resy=" "` qui définissent respectivement la résolution horizontale et verticale de l'image qui sera rendue. Le paramètre `type=" "` permet de déterminer le mode de la caméra : `perspective` et `ortho` sont deux classiques qui ne dépayseront pas les fans de Blender, mais il est également possible de compter

sur les modes `spherical` (pour un rendu panoramique sur 360°) et `lightprobe` (pour obtenir l'affichage rond typique des *light probes* utilisés dans le cas d'illumination par images `hdr`).

L'orientation de la caméra en elle-même est définie par trois points :

- `from` : dont les coordonnées `x`, `y` et `z` définissent le point-origine de la caméra ;
- `to` : dont les coordonnées `x`, `y` et `z` définissent le point visé par la caméra ;
- `up` : dont les coordonnées x, y et z définissent un point représentant le « haut » de la caméra ; la manipulation de cette coordonnée est très pratique pour faire « tourner » la caméra sur elle-même.

C'est également dans ce bloc que l'effet de flou focal est mis en place grâce aux paramètres `dof_distance=" "` et `aperture=" "`. Le flou est mis en place sur chaque pixel à l'aide d'une sous-géométrie qu'il est possible de déterminer. Il s'agit du `bokeh`, mais nous y reviendrons dans un futur proche.

4.3 Le bloc render

C'est ici que la qualité d'un rendu va se jouer. Tout d'abord, au niveau de la profondeur de récursivité des rayons lancés par le moteur, au travers du paramètre `raydepth=" "`. Plus sa valeur sera élevée, plus les inter-réflexions (objets réfléchissants, de type miroir) ou les réfractions (objets transparents, de type verre) seront rendues fidèlement.

Traditionnellement, une image est rendue pixel par pixel, mais cette technique a tendance à produire des objets aux bords crénelés, très visibles et d'un impact désastreux sur le photoréalisme. L'anti-crénelage (ou *anti-aliasing* ou AA) est une technique qui consiste à sur-échantillonner les pixels rendus avec les couleurs des pixels voisins, ce qui a pour effet d'atténuer le crénelage et restituer des bords doux. Activer l'anti-crénelage nécessite de donner une valeur non nulle au paramètre `AA_passes=" "` ; bien sûr, plus cette valeur sera élevée, plus de fois l'anti-crénelage sera répété. À chaque passe, `AA_minsamples=" "` détermine le nombre de sur-échantillon appliqué au pixel.

Vous noterez que c'est dans ce bloc qu'un chemin (ainsi qu'un nom) est attribué à l'image rendue par Yafray, grâce au paramètre `outfile value=" "`.

4.4 L'arrière-plan

L'arrière-plan est défini par un bloc de type `background`, mais est concrètement mis en place, dans le bloc `camera` actif, au travers d'un paramètre `background_name=" "` citant nommément l'arrière-plan à utiliser, parmi tous ceux potentiellement détaillés.

Il existe plusieurs types d'arrière-plan, bien évidement mis en place par un paramètre `type=" "`. Parmi les possibilités, nous retrouvons l'`image` (nécessitant un attribut `<filename value=" "/>` indiquant le chemin vers l'image `tga` ou `jpeg` appropriée), l'arrière-plan `constant` (dont la couleur est déterminée par un attribut `<color r=" " g=" " b=" "/>`), l'image `HDRI` (nécessitant également un attribut `<filename value=" "/>` indiquant le chemin vers le fichier approprié ainsi qu'un paramètre `mapping = ""` indiquant un fichier de type `probe` ou `spherical`).

4.5 Le bloc light

Il existe différents types de lampes, mais les deux plus communes sont celles permettant de produire des ombres dures (`type="pointlight"`) ou douces (`type="spherelight"`). Une fonctionnalité intéressante est la possibilité d'adjoindre à la source lumineuse une lueur (*glow*) grâce à la combinaison de paramètres `glow_intensity=" "` (qui détermine la puissance de l'effet), `glow_offset=" "` (qui gère la profondeur de couleur de la lueur) et `glow_type=" "` (qui décide du modèle de lueur à employer). Cette lueur permet à la lampe de devenir « visible » pour la caméra ainsi que pour les objets transparents. Toutefois, la lueur n'est pas visible sur un simple *background* (par exemple `<background type="constant" name="world_background" >`) : c'est une erreur courante des utilisateurs novices de Yafray, la parade étant simplement de disposer des objets (murs, plafonds et autres) en arrière-plan de la lampe.

4.6 Le bloc object

Yafray ne supporte que les maillages, aussi n'existe-t-il aucune primitive comme c'est par exemple le cas pour Povray. Il faudrait donc certainement être un brin masochiste pour essayer de décrire des figures complexes à partir d'un éditeur de texte directement dans le fichier XML : ce format, ainsi que le SDL de Yafray, n'ont pas été conçus pour cela. Yafray est et ne restera jamais rien d'autre qu'un moteur de rendu, un compagnon pour votre outil de modélisation préféré, quel qu'il soit.

Directement dans la balise `object`, il est possible de spécifier si l'objet émet des ombres ou non (`shadow=" "`) ainsi que le nom du shader (qu'il conviendra d'avoir défini auparavant) qui lui sera attribué.

Le maillage est lui-même décrit à l'intérieur du bloc `object`, entre deux balises `<mesh>` et `</mesh>`. Le paramètre `autosmooth=" "` est utile pour spécifier si les facettes de l'objet sont visibles, ou si le *shading* des facettes est lissé de façon à ce que l'ombrage de l'objet soit parfaitement continu.

Enfin, le bloc `mesh` contient lui aussi deux blocs : un bloc `points` et un bloc `faces`, qui permettent de définir la géométrie du maillage. Il est important de noter que Yafray ne supporte que les triangles, mais aucunement les quadrangles. Lors de l'export du cube par défaut de Blender, par exemple, vous obtiendrez 12 facettes au lieu des 6 que vous seriez normalement en droit d'attendre pour un cube (2 triangles par facette) : pas d'inquiétude, c'est normal ! La première ligne

précise les points à considérer pour la construction de la facette triangulaire, la seconde précise l'origine Orco pour la texture, la troisième, à nouveau les points de construction, et ainsi de suite. Enfin, pour chaque facette, il vous est possible de définir un shader à appliquer (`shader_name=" "`) ; comme précédemment, le shader en question devra avoir été défini au préalable au sein d'un bloc `shader`.

4.7 Le bloc shader

Ce bloc a une structure extrêmement simple, mais il fait partie de ceux qui sont les plus riches en paramètres et options. Outre les paramètres typiques (`type=" "` et `name=" "`) que l'on retrouve dans la balise ouvrante, le bloc `shader` contient un sous-bloc `attributes`, où les choses les plus intéressantes se jouent. C'est en effet ici que les différents paramètres du shader sont spécifiés, et l'utilisateur régulier de Blender n'aura pas grand mal à reconnaître et retenir les principales options.

Les trois couleurs de base, `color`, `specular_color` et `mirror_color`, sont chacune définies par les classiques composantes `r=" "`, `g=" "` et `b=" "`. Le paramètre *Ref* de Blender trouve son équivalent dans l'attribut `diffuse_reflect value=" "`, tandis que le lien avec les autres attributs est souvent très facile : `specular_amount value=" "` pour *Spec* ; `hard value=" "`, `alpha value=" "` et `emit value=" "` pour les propriétés homonymes de Blender, etc.

Les types de shaders, qu'ils soient de type diffus ou spéculaire, sont déterminés par les attributs `diffuse_brdf=" "` et `specular_brdf=" "`.

Conclusion

Nous n'avons bien sûr fait que survoler la syntaxe particulière de Yafray, mais nous en avons désormais une vision d'ensemble, et c'est là tout le but de cet article. Pour fonctionner, Yafray repose donc sur un SDL qui lui est propre. Mais contrairement au SDL de Povray, il n'est pas prévu pour composer des scènes entières avec seulement un éditeur de texte. Un modeleur solide, nanti de la possibilité d'exporter les objets au format XML de Yafray est donc d'une rigoureuse nécessité. L'éditeur de texte devra être robuste, également, car si l'on considère le nombre de lignes nécessaires à la description d'un simple cube, on peut imaginer le résultat sur une scène entière composée de plusieurs entités organiques ! Les fichiers XML risquent alors de prendre des proportions déraisonnables. Apprendre à naviguer dans de tels fichiers éléphantesques est alors une absolue nécessité si vous souhaitez tirer parti des capacités remarquables de Yafray, et c'est exactement ce qui a motivé l'étude du fichier XML correspondant à la (très simple !) scène par défaut de Blender : si vous avez saisi sa structure, vous serez armé pour vous attaquer à n'importe quelle scène, pratiquement quelle que soit sa taille !

A vos éditeurs de texte et à bientôt pour l'exploration des talents cachés de Yafray !!!

Olivier Saraja,

olivier.saraja@linuxgraphic.org

LIENS

- Le site de Yafray : www.yafray.org [en]
- Les forums consacrés à Yafray : www.yafray.org/forum/index.php [en]
- La documentation de Yafray : wiki.yafray.org/bin/view.pl/UserDoc/WebHome [en]
- La foire aux questions (FAQ) de Yafray : wiki.yafray.org/bin/view.pl/UserDoc/FaqEng [en]
- De l'aide en français ? Les forums de Linuxgraphic : www.linuxgraphic.org/forums/ [fr]
- Le site de Blender : www.blender.org [en]
- Le site de Wings3D : www.wings3d.com [en]

Lutter contre le spam avec Postfix

Xavier Guimard

EN DEUX MOTS **Dans les articles précédents, nous avons abordé la sécurisation interne d'un système de messagerie avec Postfix ainsi que la problématique du routage du courrier. Nous en venons ici à la lutte contre le courrier indésirable.**

Cet article peut être lu indépendamment des deux précédents. Toutefois, il fait appel à des notions déjà présentées précédemment.

Classification du courrier indésirable

Pour bien appréhender la lutte contre le courrier indésirable, nous allons en établir un classement. Du point du vue de la provenance, on peut établir trois catégories :

- interne : courrier indésirable provenant de l'intérieur du réseau. Il peut s'agir du courrier généré par un virus ou tout simplement d'un utilisateur indélicat ;
- externe : le traditionnel flot d'attaques par virus ou spam ;
- fausses notifications : certains robots utilisent de fausses adresses sources calculées à partir de dictionnaires ou de carnets d'adresses piratés. Si l'adresse source du message non désiré semble provenir d'un de nos domaines, et suivant la configuration du site attaqué, nous pouvons recevoir un avis de non remise.

Les catégories interne et externe peuvent, elles aussi, être subdivisées : le client IP peut être un virus ou un serveur SMTP licite ayant accepté un message indésirable.

En termes d'effet, la classification est assez simple :

- spam : les publicités ;
- exécutables dangereux standards : la catégorie des virus, chevaux de Troie, et autres codes malveillants existant sur le marché ;
- exécutables dangereux spécifiques : nous classons ici les éventuels codes malveillants qui auraient été écrits spécialement pour nous attaquer dans le cadre par exemple de l'espionnage industriel.

Insertion d'un dispositif d'examen du contenu dans Postfix

Postfix est avant tout un MTA, il n'a donc pas été écrit pour rechercher les virus ou le spam. En revanche, sa structure modulaire lui permet d'accueillir, dans son flux de message, les logiciels spécialisés dans le filtrage.

Il existe plusieurs façons d'intégrer un dispositif d'examen du contenu dans Postfix. Commençons par la moins bonne :

- Insertion du dispositif en coupure sur le réseau : dans ce cas, le courrier arrive directement sur l'équipement ou le logiciel qui le traite et le renvoie à Postfix. L'inconvénient majeur de ce dispositif est que l'attaquant est directement renseigné sur l'équipement utilisé (par la bannière par exemple). D'autre part, comme tous ces équipements sont plus ou moins vulnérables aux attaques de type déni de service (par débordement de la mémoire disponible par exemple), nous risquons une indisponibilité du service de messagerie. L'attaquant peut alors chercher à nous faire passer en mode dégradé, car, en général, quand on ne comprend pas pourquoi ça ne marche pas, on débranche les équipements de sécurité :-(
- Insertion d'un dispositif dans le système Postfix traitant à la volée le courrier entrant. Postfix est ainsi apte à rejeter le courrier indésirable pendant la livraison ce qui évite la génération des éventuels avis de non-remise, mais risque de ralentir le traitement des messages. Dans un site soumis à une forte charge, cette méthode est déconseillée pour un traitement lourd, car le client connecté risque de ne pas recevoir assez vite la réponse de bonne réception et tentera de multiples fois la même livraison. Il existe deux façons de faire cette intégration : en proxy SMTP ou avec le protocole MILTER [1] de la communauté Sendmail.
- Insertion d'un dispositif après mise en file d'attente : Postfix accepte le courrier sans effectuer ce contrôle, puis présente les messages à un rythme contrôlable au dispositif de filtrage. Cette méthode présente les caractéristiques inverses de la précédente : robustesse, mais nécessité de générer les éventuels avis de non remise.

Postfix dispose également d'un dispositif interne de filtrage assez sommaire : l'examen des en-têtes et du corps du message. Ce dernier présente les limites suivantes :

- Certains sont tentés d'écrire de nombreuses règles de filtrage en utilisant ces capacités : ça ne fera jamais de Postfix un logiciel anti-spam et la rapidité de traitement des messages risque d'en pâtir.
- L'examen du corps qui traite le message se fait ligne par ligne sans décodage : la recherche d'un mot ne peut être effectuée dans une pièce jointe, ni dans une partie de message encodée (en base 64 par exemple).

En revanche, ce dispositif peut faciliter l'intégration d'un dispositif comme nous le verrons par la suite. Ces généralités étant dites, passons à la technique :

Insertion d'un dispositif en coupure

Dans une telle configuration, Postfix ne peut plus distinguer le courrier entrant du courrier sortant. Le contrôle anti-relais est alors délégué au dispositif amont (sauf s'il est compatible XFORWARD). La configuration de Postfix est alors extrêmement simple, il se contente d'assurer le routage. Si les deux logiciels se trouvent sur le même serveur, on change le port en écoute de Postfix (pour le laisser au dispositif de filtrage) en remplaçant dans le fichier `master.cf` la ligne :

```
smtp  inet  n  -  -  -  -  smtpd
```

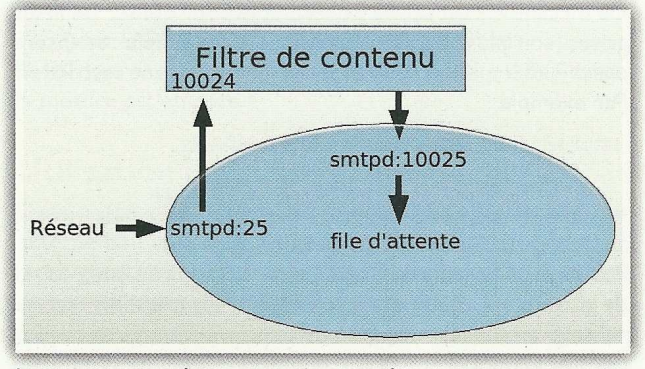
par la ligne :

```
127.0.0.1:11000  inet  n  -  -  -  -  smtpd
```

On met ainsi le serveur `smtpd` en écoute sur la boucle locale seulement sur le port 11000, port sur lequel le dispositif de filtrage doit renvoyer le courrier.

Insertion d'un proxy SMTP

Le principe est ici d'insérer le dispositif entre deux démons `smtpd` de Postfix :



Le premier en écoute sur le port 25 transmet toutes les commandes au dispositif configuré pour renvoyer le courrier au deuxième serveur smtpd en écoute sur le port 10026. Le dispositif de filtrage doit être à même de comprendre toutes les commandes SMTP reçues par Postfix et doit ainsi être compatible ESMTP.

Ce qui donne dans le fichier `master.cf` :

```
smtp  inet  n  -  n  -  20  smtpd
  -o smtpd_proxy_filter=machine:10025
  -o smtpd_client_connection_count_limit=10
#
# Serveur SMTP après-filtrage. Reçoit le courrier du filtre de
# contenu sur le port 10026.
#
:10026  inet  n  -  n  -  -  smtpd
  -o smtpd_authorized_xforward_hosts=machine
  -o smtpd_client_restrictions=
  -o smtpd_helo_restrictions=
  -o smtpd_sender_restrictions=permit_mynetworks,reject
  -o smtpd_recipient_restrictions=permit_mynetworks,reject
  -o smtpd_data_restrictions=
  -o mynetworks=machine/32
  -o receive_override_options=no_header_body_checks
```

Explications :

- On configure le premier `smtpd` pour utiliser le proxy smtpd **[2]**, et on limite sa capacité pour éviter un effondrement du dispositif de filtrage.
- On retire au deuxième `smtpd` toute charge de filtrage et de contrôle (déjà assuré par le premier). Il se contente de n'accepter du courrier qu'en provenance du filtre et on lui ordonne d'accepter les commandes XFORWARD.

Insertion d'un filtre MILTER

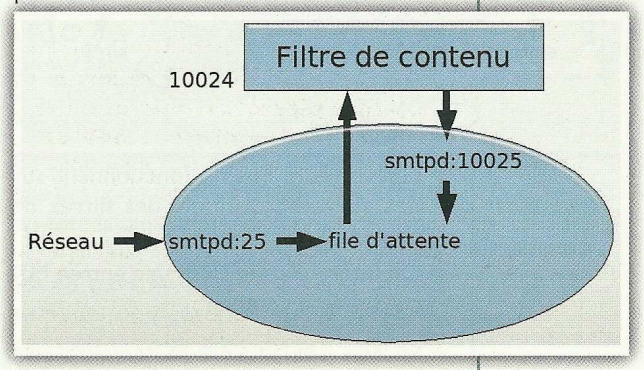
L'emploi du protocole MILTER **[1]** ne nécessite pas de deuxième démon `smtpd`. Il suffit d'indiquer dans `main.cf` la liste des filtres MILTER à utiliser :

```
cleanup_milters = inet:localhost:11025, ...
```

Ce mécanisme n'est disponible qu'à partir de la version 2.3 de Postfix.

Insertion d'un filtre après mise en file d'attente

Ce dispositif est assez similaire au mode proxy si ce n'est qu'on utilise la directive `content_filter` en lieu et place de la directive `smtpd_proxy_filter` **[2]**. On peut également utiliser dans cette configuration un filtre qui reçoit et/ou soumet le courrier par d'autres protocoles.



Exemple : notre filtre reçoit les messages par un *pipe* Unix et les envoie en utilisant la commande `sendmail` de Postfix :

- Le fichier `master.cf` contient :

```
filtre  unix  -  n  n  -  10  pipe
  flags=Rq user=filter argv=/rep/script -f ${sender} -- ${recipient}
```

- La directive de filtrage appelle notre filtre :

```
content_filter=filtre:rien.
```

- Le filtre lit le message sur l'entrée standard et appelle `sendmail` :

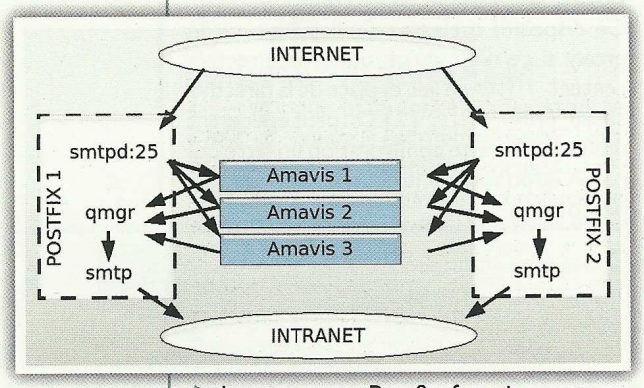
```
#!/bin/sh
# Nettoyage lors en sortant ou lors d'une interruption
trap "rm -f in.$$" 0 1 2 3 15
# Démarrage du processus.
cd /tmp
cat >in.$$
# filtrage à écrire sur le fichier in.$$
# Renvoi
sendmail -G -i "$@" <in.$$
# Renvoi du code de retour de sendmail
exit $?
```

Robustesse

Quel que soit le mode choisi, un processus de filtrage défaillant peut rendre indisponible le service de messagerie : s'il fonctionne sur la même machine, il peut absorber toute la mémoire disponible ou tout simplement ne plus répondre.

On peut contrôler la charge du filtre en limitant le nombre de livraisons parallèles comme nous l'avons déjà vu, mais ça ne nous préserve pas d'un dysfonctionnement plus grave lié à un virus non supporté par exemple.

En utilisant le mécanisme de répartition de charge basé sur les MX présentés précédemment dans cette série d'articles, on peut créer un système rustique :



- Les serveurs Postfix fonctionnent sur des serveurs distincts des filtres de contenu.
- On insère dans le DNS une entrée MX par filtre de contenu :

```
mx-filtrage  IN  CNAME  filtre1
mx-filtrage  IN  CNAME  filtre2
```

- On crée un transporteur smtp particulier dans `master.cf` pour ce filtre pour contrôler la charge envoyée :

```
filtre  unix  -  -  -  -  -  smtp
  -o smtp_send_xforward_command=yes
```

et on limite son usage dans `main.cf` :

- On utilise ce champ dans la directive

```
content_filter (ou smtpd_proxy_filter) : content_
filter = filtre:mx-filtrage:10024.
```

- Si le filtre est capable d'utiliser le même mécanisme pour renvoyer le courrier à la deuxième instance du serveur `smtpd`, on crée les entrées MX correspondantes, sinon on fait la même chose avec les enregistrements de type A (*round-robbin*) :

```
routage  IN  A  postfix1
routage  IN  A  postfix2
```

Le filtre renvoie alors le courrier vers `routage:10025`. L'inconvénient des enregistrements A est que si l'un des Postfix est injoignable, le filtre peut être redirigé lors des tentatives suivantes vers le même serveur. On risque ainsi de perdre du courrier, même si cette probabilité est assez faible.

L'emploi d'Amavis **[3]** amène également une certaine rusticité, puisque ce logiciel sert d'interface entre Postfix, l'anti-spam et les anti-virus. On peut alors également utiliser des anti-virus de secours en cas de défaillance des premiers. Il ne protège toutefois pas des problèmes de saturation de la mémoire. Il s'utilise avec `smtpd_proxy_filter` ou `content_filter`. Par défaut, il est en écoute sur le port 10024 et renvoie son courrier vers le port 10025.

Autres mécanismes disponibles dans Postfix

Délégation de la politique d'accès

Postfix dispose d'un autre mécanisme permettant d'insérer un filtre extérieur : la délégation de politique d'accès **[5]**. Il ne s'agit pas ici pour le filtre d'examiner le contenu du message, mais simplement, à partir des données de connexion (adresse IP, expéditeur, nom communiqué dans le HELO...), de retourner une action de type table d'accès **[4]** (ex : `action=reject`). Ce mécanisme est utilisé par exemple pour utiliser les listes grises (voir plus bas). Son emploi est assez simple : on insère `check_policy_service inet:host:port` dans une restriction. Par exemple :

```
smtpd_recipient_restrictions = permit_mynetworks,
        reject_unauth_destination,
        check_policy_service inet:127.0.0.1:60000
```

Listes noires en temps réel (RBL)

Postfix peut également intégrer la consultation de listes noires de sites mises à jour en temps réel. Il faut toutefois bien se renseigner sur les mécanismes utilisés pour mettre à jour ces listes avant d'en choisir une pour éviter de rejeter un grand FAI sous prétexte qu'un de ses clients fait l'objet d'une plainte. Comme pour la délégation de politique d'accès, leur emploi s'effectue au travers des restrictions. Exemple :

```
smtpd_recipient_restrictions = permit_mynetworks,
            reject_unauth_destination,
            reject_rbl_client relays.ordb.org,
```

La liste publiée par ORDB n'est pas très efficace, mais n'amène aucune erreur, car le site vérifie que la machine dénoncée est réellement un relais ouvert avant de la mettre en liste noire.

Lutter contre le courrier indésirable avec Postfix

La lutte contre le courrier indésirable n'est pas un problème facile. Outre la complexité des technologies utilisées, il est très difficile pour un administrateur d'un grand réseau d'appliquer une politique de filtrage uniforme et qui convienne à tous les utilisateurs. Dans certains cas, il est même inadmissible de prendre le risque de bloquer un message licite. A l'inverse, les utilisateurs acceptent parfois assez difficilement de recevoir 15 publicités par jour en anglais pour des produits pharmaceutiques. Si la technologie anti-spam n'est pas une réponse à tout, certains mécanismes basés sur des règles logiques peuvent contribuer à éliminer le courrier indésirable sans aucun doute.

Interdiction des exécutables malveillants

Un anti-virus mis à jour en temps réel élimine à lui tout seul la menace que nous avons appelée « exécutables dangereux standards » à ceci près qu'il n'est pas toujours en mesure d'éliminer à temps le dernier sorti des virus.

L'élimination des exécutables malveillants spécifiques ou très récemment sortis ne peut passer que par une interdiction formelle de tout ce qui présente un danger pour le client. Le logiciel Amavis **[3]** est en mesure d'éliminer les pièces jointes par extension (.exe,...) ou par `Content-Type`. Cette mesure ne suffit pas à protéger les utilisateurs, car il existe d'autres moyens de lancer du code malveillant (certaines balises HTML, les Javascripts, les XPI...). Pour avoir la politique la plus fine possible, il faut connaître les logiciels clients utilisés et suivre les menaces pour adapter ce filtrage.

Menace interne

Une mesure simple pour limiter les éventuelles nuisances internes consiste à exiger une authentification interne sur TLS (voir LM N° 85). Ça n'élimine pas tout, mais couplée à l'anti-virus et aux autres éléments de la politique de filtrage, la mesure est efficace.

Utilisation des listes grises

Dans notre classification, nous avons distingué le courrier externe issu d'un robot ou déjà accepté par un serveur de messagerie. Le mécanisme le plus efficace pour lutter contre les robots consiste à utiliser les listes grises. Le principe en est assez simple. On rejette toute première requête d'un client avec un code d'erreur temporaire (4xx). Si le serveur présente de nouveau son message, on peut supposer que le client est un vrai serveur de messagerie.

Comme nous l'avons indiqué plus haut, le mécanisme le plus approprié dans ce cas est l'emploi de la délégation de politique d'accès.

Sous Debian, l'installation d'un dispositif de liste grise est très simple (comme d'habitude ;-)) **[6]** :

```
apt-get install postgrey
```

Le démon est alors installé et en écoute sur le port 60000. Il ne reste plus qu'à indiquer à Postfix de l'utiliser comme au paragraphe « délégation de la politique d'accès ».

Le problème avec ces listes est que certains sites supportent assez mal le rejet et ne représentent le message que beaucoup plus tard ou plus du tout. Il faut donc entretenir une liste blanche des serveurs susceptibles.

Filtrage intégré

On peut faire beaucoup de contrôles avec Postfix en vérifiant la validité de tous les champs, en filtrant les en-têtes,... il faut toutefois bien expérimenter ces filtres pour mesurer leur impact. Le mot clef `warn_if_reject` **[7]**, qu'on peut utiliser devant n'importe quelle restriction, permet d'enregistrer le rejet dans les journaux sans qu'il soit effectif et peut alors vous rendre de grands services.

Examen des commandes EHLO/HELO

En théorie, le nom d'hôte passé dans la commande HELO devrait correspondre au nom DNS du serveur, mais comme indiqué dans la RFC 2505 **[8]**, il n'est pas conseillé d'examiner de manière trop stricte ce nom. La restriction suivante rejette ainsi la plupart des robots, mais malheureusement aussi quelques grands FAI mal administrés :

```
smtpd_helo_restrictions = permit_mynetworks,
            reject_invalid_hostname,
            reject_unknown_hostname
```

A utiliser donc avec précaution. Certains logiciels utilisés pour la délégation de politique d'accès tels `policyd` **[9]** proposent des listes noires de noms d'hôtes utilisés dans les commandes HELO.

Rejet des fausses notifications

L'avis de non-remise contient généralement les en-têtes du message d'origine. Les robots, cherchant généralement à singer le site apparemment source, insèrent dans les en-têtes ou dans le nom communiqué avec la commande HELO des termes semblant indiquer la provenance de notre site. L'astuce proposée ici consiste à rejeter les messages contenant des en-têtes manifestement falsifiés.

En supposant que nos serveurs n'utilisent que des noms de type `mail#.domaine.com`, un en-tête faisant référence à une commande `HELO domaine.com` doit être rejeté. De même, les en-têtes `Message-Id` générés par nos serveurs ne peuvent contenir `@domain.com`, mais se terminent par `@mail#.domaine.com` **[10]** :

```
# /etc/postfix/main.cf
header_checks = pcre:/etc/postfix/entetes
body_checks

# /etc/postfix/entetes
/^Received: +from +(domaine\.com) +/
  reject forged client name in Received:
header: $1
/^Received: +from +[^ ]+ +\(([^ ]+
+[he]+lo=|[he]+lo +)(domaine\.com)\)/
  reject forged client name in Received:
header: $2
/^Message-ID:.*@(domaine\.com)/
  reject forged domain name in Message-ID:
header: $1

/etc/postfix/body_checks:
/^[> ]*Received: +from +(domaine\.com) /
  reject forged client name in Received:
header: $1
/^[> ]*Received: +from +[^ ]+ +\(([^ ]+
+[he]+lo=|[he]+lo +)(domaine\.com)\)/
  reject forged client name in Received:
header: $2
/^[> ]*Message-ID:.*@(domaine\.com)/
  reject forged domain name in Message-ID:
header: $1
```

Rejet des adresses de destination aléatoires

Le problème des adresses de destination aléatoires est que les messages peuvent être acceptés par le serveur SMTP et c'est seulement à la livraison finale que le système découvre que le destinataire n'existe pas. Le système génère alors un avis de non-remise qui va générer de nouvelles fausses notifications vers l'extérieur. Pour ne pas cotiser à ce désordre, il est important de rejeter ces messages dès la réception de la commande `RCPT TO`.

Par défaut, Postfix renseigne la directive `local_recipient_maps` pour n'accepter que les utilisateurs disposant d'un compte Unix ou renseignés dans les alias. En cas d'utilisation d'un autre dispositif de gestion des boîtes aux lettres, le simple fait de renseigner ce paramètre (par une table LDAP par exemple) sur le serveur recevant le courrier extérieur élimine le problème des adresses aléatoires.

Pour les domaines relayés, Postfix dispose d'un mécanisme équivalent, mais évidemment non renseigné par défaut : `relay_recipient_maps`.

Les tables passées en paramètre à ces deux directives peuvent renvoyer n'importe quelle valeur. Postfix vérifie simplement que le destinataire existe. On peut ainsi utiliser des tables conçues pour le routage par exemple.

Traitement du spam résiduel

Une fois toutes ces mesures en place, il nous reste le plus difficile à éliminer : le spam bien conçu et déjà accepté par une passerelle de messagerie externe. A ce niveau, ce n'est plus l'affaire de Postfix, mais bien d'un logiciel spécialisé. Amavis **[3]**, nous permet une fois de plus de connecter très simplement le très célèbre Spamassassin : tout est prêt dans les fichiers de configuration d'Amavis, il n'y a plus qu'à décommenter la ligne.

Nous ne rentrerons pas ici dans la configuration de Spamassassin qui pourrait, à elle seule, constituer une série d'articles. Signalons juste quelques-uns des mécanismes utilisés par Spamassassin :

- Filtres bayesiens et heuristiques : ils donnent des résultats approximativement sûrs, mais peuvent rejeter du courrier licite si la barre de notation est fixée trop bas. Ces filtres sont en revanche très efficaces sur le poste client (comme le gestionnaire des indésirables de Thunderbird), car la politique est individualisée.
- Listes type `razor` : consultation d'une base de données pour voir si le message est déjà connu dans la base. Ces listes donnent de très bons résultats, sauf au tout début de la propagation d'un spam.

Conclusion

La lutte contre le courrier indésirable est trop complexe pour être confiée à un seul mécanisme dans un grand réseau (à moins qu'on en accepte les risques). J'espère vous avoir apporté dans cet article quelques éléments d'éclairage sur les possibilités intrinsèques de Postfix et sa facilité à accueillir d'autres dispositifs. J'utilise personnellement ce mécanisme pour insérer de petits programmes Perl effectuant divers traitements sur les messages, comme des extractions vers des bases de données ou encore des contrôles de cohérence entre les adresses d'enveloppe et celles renseignées dans les en-têtes. Peut-être dans un prochain article ?

Xavier Guimard

http://x.guimard.free.fr/postfix/
http://postfix.traduc.org/

Responsable de la traduction de la documentation de Postfix en Français. Co-développeur du projet Lemonldap. Responsable de l'architecture des systèmes d'information de la gendarmerie nationale.

NOTES

- **[1]** Quelques explications sur le protocole MILTER et son intégration dans Postfix : **http://postfix.traduc.org/index.php/MILTER_README.html**
- **[2]** Les directives `content_filter` et `smtpd_proxy_filter` peuvent également être insérées dans le fichier `main.cf` et non comme paramètre du premier `smtpd`. Dans ce cas, elles devront être redéfinies dans le deuxième `smtpd` pour éviter un bouclage du courrier en ajoutant les options :
```
-o smtpd_proxy_filter=
-o content_filter=
```
- **[3]** Amavis est disponible sur : **http://www.amavis.org/**
- **[4]** Les actions possibles dans une table d'accès sont indiquées dans la page de manuel access(5) ou sur la page : **http://www.postfix.org/access.5.html**
- **[5]** **http://postfix.traduc.org/index.php/SMTPD_POLICY_README.html**
- **[6]** Sur Debian Sarge, il faut utiliser les portages du site **http://www.backports.org** pour obtenir `postgrey`.
- **[7]** **http://postfix.traduc.org/index.php/postconf.5.html#warn_if_reject**
- **[8]** La RFC 2505 contient des recommandations pour lutter contre le spam au niveau des MTA : **http://rfc.net/rfc2505.html**
- **[9]** Policyd implémente à la fois les listes grises et d'autres mécanismes de lutte contre le spam : **http://policyd.sourceforge.net/**
- **[10]** Par défaut, `header_checks` s'applique aux en-têtes du message ainsi qu'aux en-têtes des messages en pièces jointes. Ce mécanisme est mis en place par les valeurs par défaut suivantes :
```
nested_header_checks = $header_checks
mime_header_checks = $header_checks
```
Si vous souhaitez accueillir le courrier de personnes transférant de faux messages, il faut indiquer à Postfix que `header_checks` ne doit pas être utilisé en dehors des en-têtes du message en renseignant ces deux paramètres par d'autres tables ou rien si vous ne souhaitez aucun filtrage sur les en-têtes des pièces jointes.

Xen et l'optimisation d'espace disque

François Donzé

EN DEUX MOTS **Comment minimiser le stockage utilisé par les systèmes invités ?**

Xen, le moniteur de machines para-virtuelles du monde libre permet la cohabitation de plusieurs instances de systèmes d'exploitation dans une même machine physique. Grâce aux techniques de copie-sur-écriture (Copy-On-Write ou COW), il est possible de partager les exécutables d'un système d'exploitation par plusieurs systèmes invités. Le COW permet aussi de replacer une machine virtuelle dans un état connu, de manière quasi instantanée.

Après quelques rappels sur Xen [1] et une introduction rapide des techniques de copie-sur-écriture [2], nous proposons de démarrer plusieurs machines virtuelles à partir d'une copie unique d'un système RHEL5. La structure des médias RHEL5 ayant complètement changée, nous détaillerons quelques différences notables avec les versions précédentes de cette distribution. Nous montrerons aussi comment il est possible d'exécuter une opération de type « *undo* » permettant de revenir rapidement à un état connu d'un système invité.

Xen : quelques rappels

Xen est un Moniteur de Machines Virtuelles (VMM) très largement décrit dans *GNU/Linux Magazine* 85 et 87. Il est comparable à des produits propriétaires comme l'ESX de VMware ou l'IntegrityVM de HP. Toutefois, il est différent dans le sens où, sur des processeurs n'ayant pas de technologie spécifique de virtualisation, les machines virtuelles invitées ont besoin de modifications dans leur noyau. C'est pour cela qu'on emploie le qualificatif de « paravirtualisation », quand on parle de Xen.

Xen se démarque aussi des autres produits de virtualisation par sa terminologie. Aux synonymes « machine virtuelle » et « système invité » s'ajoute le terme « domaine », qui n'est autre qu'une machine virtuelle instanciée. Le démarrage d'un système Xen est automatiquement suivi de la création d'un premier domaine appelé « domaine0 ». Du point de vue de l'utilisateur, ce domaine est similaire à un système Linux habituel. Pourtant, il est particulier, car il contrôlera les autres domaines. Cette fonction de contrôle est assurée par le démon `xend` qui peut communiquer avec les autres domaines via des structures internes à Xen.

Les autres domaines, aussi appelés « domaines utilisateurs » ou *domaineU* sont instanciés (créés) par le domaine0 à partir d'un fichier de configuration fournissant de multiples informations. Par exemple, on y trouve le nom du domaine, la configuration réseau, l'endroit et le type du *Virtual Block Device* (VBD) contenant le système d'exploitation. Les VBD présentables aux domainesU peuvent être des fichiers, des disques, des partitions physiques locales ou distantes (SAN), des volumes logiques ou encore, à partir de Xen 3.0.3, des images au format QCoW (*Qemu Copy-on-Write*).

Dans la partie pratique de cet article, nous préparerons des volumes logiques situés sur un disque physique local, avant de les présenter aux différents domaines. Cette préparation se fera en étant connecté au domaine0.

Gestionnaires de volumes logiques

La gestion des espaces de stockage utilisant les volumes logiques (*Linux Magazine* 53) est une méthode logicielle offrant, entre autres, des fonctionnalités de haute disponibilité (RAID). Le gestionnaire de volumes privilégié dans Linux est LVM (*Logical Volume Manager*) principalement développé par Sistina [5]. Il en existe d'autres comme EVMS [6], développé par IBM.

Brièvement, les gestionnaires de volumes logiques permettent de regrouper des disques, des partitions physiques ou des unités logiques (LUN). Dans ces groupes, on aura la possibilité de créer des espaces de stockage appelés « volumes ». Ces volumes pourront être étalés (ou non) sur plusieurs disques physiques (*striping*) ou encore placés en mode miroir. Ensuite, il suffira de transformer ces volumes en systèmes de fichiers (ext2/3, reiserfs, JFS ou encore XFS) avant de les monter dans l'arborescence.

La fonctionnalité qui nous intéresse dans ces gestionnaires de volumes est celle des volumes logiques virtuels encore appelés « clones persistants » ou « *snapshots* ». Le snapshot d'un volume existant n'est autre qu'une « photo » d'un volume logique. Le terme « photo » est approprié puisqu'il est possible de voir et même d'accéder à tous les fichiers à l'instant de la prise de la photo, et à cet instant uniquement, même si les fichiers ont été modifiés depuis cette prise. Les snapshots sont souvent utilisés pour effectuer des sauvegardes de systèmes en état de marche. Cela permet d'être sûr de sauvegarder des données statiques.

Comme les snapshots ne sont pas une copie de données, mais une liste de pointeurs et de blocs de stockage, ils ne prennent pas de place significative. Toutefois, les gestionnaires de volumes associent à ces listes de pointeurs et blocs, un espace physique prévu pour contenir les modifications éventuelles des fichiers référencés par le snapshot suivant la technique de copie sur écriture.

Copie-sur-écriture

La copie-sur-écriture (*copy-on-write* ou COW) est un terme générique utilisé dans différents domaines comme la gestion de la mémoire virtuelle des systèmes d'exploitation. Succinctement, un processus souhaitant modifier une ressource sans que les autres processus ayant aussi accès à cette ressource ne voient cette modification, en fera une copie privée. Si la ressource devant être modifiée n'est accessible qu'en lecture seule, le processus la recopiera dans une zone lecture-écriture afin de la modifier à son gré.

C'est une variante de ce dernier cas de figure qui nous intéresse. En effet, nous allons créer des snapshots d'un système d'exploitation complet et les présenter aux différentes VM. Toute modification de ces snapshots par la VM sera écrite dans la zone associée et réservée à cet effet.

Il existe d'autres techniques de copie-sur-écriture que les snapshots de volumes logiques, comme par exemple les disques image clairsemés de type QCoW (*Qemu Copy on Write sparse disk image*). Les disques QCoW ont un format spécifique et sont supportés comme VBD à partir de Xen 3.0.3. Les utilitaires `img2qcow`, `qcow-create` et `qcow2raw` sont fournis pour les gérer. Bien qu'on appelle ces espaces de stockage des « disques », ce sont en réalité des fichiers situés sur un système de fichier de la machine hôte et présentés comme disques aux machines virtuelles.

Outre leur caractéristique de copie-sur-écriture, ces fichiers sont « troués », clairsemés de zéros dont l'espace sur le système de fichier du système hôte n'est pas comptabilisé comme utilisé. Ainsi, seul l'espace réellement utilisé apparaît à l'utilisateur. Ce type de VBD est une alternative aux snapshots des volumes logiques.

RHEL5 : nouveau format

De la même manière que la distribution SLES10 est la première distribution de Novell incluant Xen, RedHat, pour la première fois, propose ce produit de virtualisation dans RHEL5. Ainsi, à partir d'un même média (CD ou DVD) de RHEL5, il sera possible de générer soit un serveur classique sans spécificité particulière, soit un moniteur de machines virtuelles Xen, soit un système invité Xen. La matrice des différentes possibilités est plus complexe, puisque cette nouvelle version inclut aussi des fonctions de mise en grappe (*cluster*), avec ou sans l'utilisation de système multi-chemin d'accès au stockage dénommé « *ClusterStorage* ».

Afin de permettre à l'utilisateur de sélectionner simplement les combinaisons de son choix, RedHat a complètement modifié la structure des répertoires sur les médias de cette nouvelle distribution. Très tôt dans le processus d'installation, l'utilisateur doit entrer une clé d'activation (figure 1) qui conditionnera l'affichage des différentes catégories de produits à installer.

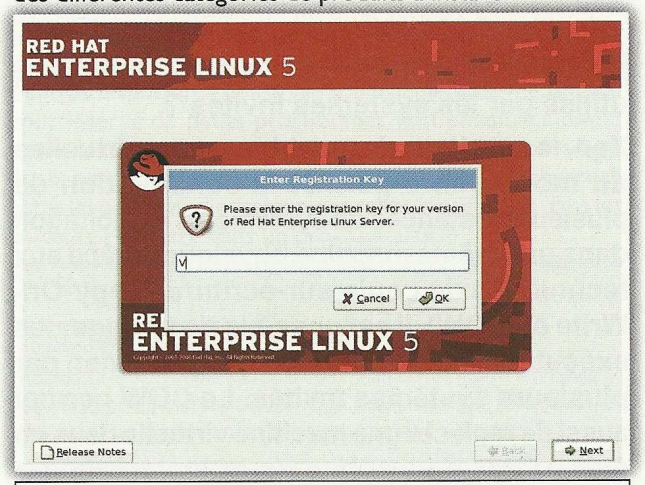


Figure 1 : Clé d'activation RHEL5 (bêta) permettant de générer un VMM Xen

Si, durant la génération d'un système hôte, cette clé d'activation contient le caractère majuscule « V » comme indiqué sur la figure 1, on trouvera plus tard la catégorie de paquets « Virtualisation » aux côtés des catégories « système de base », « langages», « applications »... En sélectionnant les paquetages de cette nouvelle catégorie, on générera un moniteur Xen (figure 2).

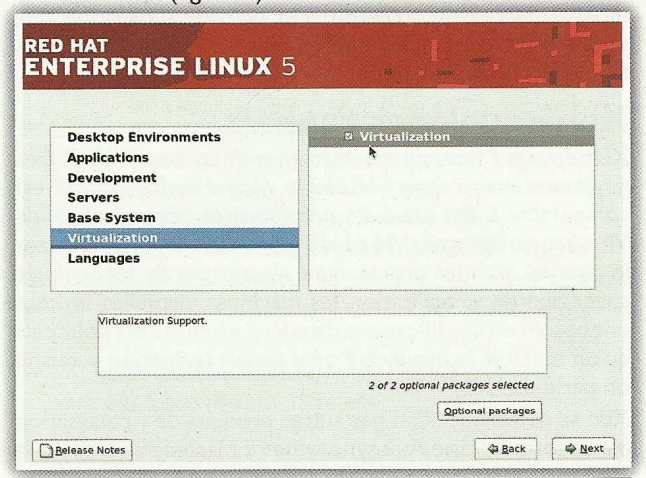


Figure 2 : Catégories contextuelles de paquetages dans RHEL5

Hormis le système de clé d'activation comportant ces caractères significatifs, le reste du processus de génération d'un système hôte de type RHEL5 est globalement semblable à celui de RHEL4. Prenons quand même soin d'activer une méthode de partage de fichier à distance comme le serveur NFS ou un serveur FTP, car nous aurons besoin de cette fonctionnalité pour la génération des systèmes invités (figure 3).

Partage des binaires : méthodologie

Considérant que notre serveur Xen est prêt à héberger des systèmes invités, nous devons maintenant identifier en son sein, un espace de stockage permettant de contenir la distribution d'un système invité. Bien qu'il soit possible d'utiliser les fichiers et exécutables du domaine0 pour constituer nos futures machines virtuelles, nous ne choisirons pas cette option. Une

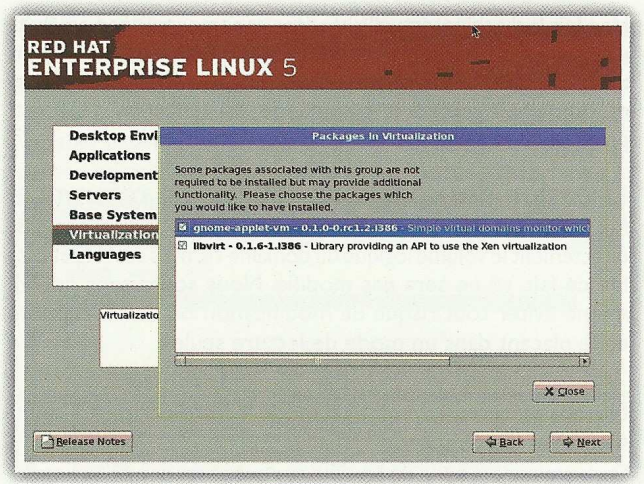


Figure 3 : Paquetages de virtualisation optionnels dans RHEL5

telle d'architecture est pourtant valide, puisque utilisée dans le CD de démonstration fourni par XenSource **[3]**. Ce *live* CD met en commun les fichiers du domaine0 et des domaines utilisateurs grâce au système de fichier *unionfs* **[4]**, qui permet la copie-sur-écriture. Ainsi, il est possible de démarrer un nombre considérable de domaines en plus du domaine0, à partir d'un CD de seulement 650 MO.

Afin de simplifier notre exercice, nous avons choisi de séparer complètement les espaces de fichiers du domaine0, et ceux de ses systèmes invités. Nous allons créer un volume logique de type LVM2 dans lequel nous générerons un système invité. Ensuite, nous créerons des snapshots de ce volume que nous présenterons en mode d'écriture, aux différentes machines virtuelles.

Le choix du gestionnaire de volumes (*volume manager*) LVM2 est naturel, puisqu'il est installé par défaut dans les systèmes RHEL, qu'il dispose de la fonctionnalité des snapshots et qu'il est aussi utilisé par défaut lors de la génération du serveur (Xen ou autre). Rien n'empêche l'utilisation d'un autre gestionnaire de volumes comme EVMS, si ce n'est qu'il nécessite un petit travail de rapatriement de paquetages non disponibles en standard dans RHEL5.

Création d'un volume logique pour le système invité de référence

Notre serveur Xen possède plusieurs disques physiques et la commande `fdisk -l` indique que le disque de 18 GO `/dev/sdd` est libre et donc utilisable dans sa totalité. Initialisons-le pour ensuite le placer dans un groupe de type LVM2 :

```
Xen# pvcreate /dev/sdd
  Physical volume "/dev/sdd" successfully created
```

Ensuite, nous créons un groupe de volumes appelé `vg` et dans lequel nous plaçons le disque précédemment formaté :

```
Xen# vgcreate vg /dev/sdd
  Volume group "vg" successfully created
```

La dernière étape consiste à constituer un volume logique assez large dans ce groupe de volumes, pour héberger un serveur RHEL5. Nous l'appellerons `vm` et lui affecterons une taille de 5 GO :

```
Xen# lvcreate -L5120 -n vm vg
  Logical volume "vm" created
```

Grâce aux trois commandes précédentes, nous disposons d'un espace de stockage de 5 GO de type LVM2, permettant d'héberger un système d'exploitation de type RHEL5. Après avoir généré un tel système dans cet espace, nous le prendrons en « photo » (*snapshot*) et l'utiliserons comme système de base pour plusieurs machines virtuelles.

Génération du système invité de référence

À partir de RHEL5-beta1 (septembre 2006), le processus de génération d'un système Xen invité diffère sensiblement de celui décrit dans la documentation de Xen 3.0 et téléchargeable chez XenSource. Cette documentation n'est en effet valable que pour RHEL4 et les versions de développement de RHEL5 d'avant septembre 2006. De ce fait, nous souhaitons détailler cette étape importante.

Tout d'abord, nous nous placerons dans un contexte défavorable où la distribution est présente sous la forme de cinq fichiers `.iso` stockés dans le répertoire `/var/ftp/kits`. Nous devons les « concaténer » dans un répertoire unique. En effet, il est nécessaire de disposer de la distribution complète et « éclatée », en un lieu unique de l'arborescence.

Le petit script suivant permet de regrouper le contenu des cinq disques en un endroit unique. Notez que le `disc1` est différent des autres, car il comprend le répertoire `images`, qui contient les amorces de démarrage et les fichiers de description de la distribution.

```
RNAME=/var/ftp/kits
DNAME=RHEL5-Server-20060927.0-i386
mkdir $RNAME/$DNAME
for i in 1 2 3 4 5 ; do
    mkdir $RNAME/disc$I
    mount -o loop=/dev/loop$i $RNAME/$DNAME-disc${i}-ftp.iso $RNAME/disc$I
    [ $i == 1 ] && cp -ax $RNAME/disc1/images $RNAME/$DNAME
    cp -ax disc${i}/{Cluster,ClusterStorage,VT,Server} $RNAME/$DNAME
    umount $RNAME/disc$I
done
```

Le cœur du script précédent est la commande `cp -ax` qui copie les répertoires `Cluster`, `ClusterStorage`, `VT` et `Server` de chacun des fichiers `.iso` montés en « *loopback* » vers le répertoire final. Le sous-répertoire `Server` contient les paquetages principaux de la distribution RHEL5. Les sous-répertoires `Cluster`, `ClusterStorage` et `VT` contiennent les paquetages correspondant aux options C, S, V pouvant être placées dans la boîte de dialogue de la clé d'activation.

Ainsi, ces quelques lignes de *shell* nous permettront de générer un système invité de référence, à partir de la distribution éclatée dans le répertoire `/var/ftp/kits/RHEL5-Server-20060912.2-i386`. Cette génération pourra être effectuée soit via la ligne de commande et le script `xenguest-install`, soit grâce à l'assistant de création de l'interface graphique `virt-manager` **[7]** intégrée à Gnome (figure 4, page suivante).

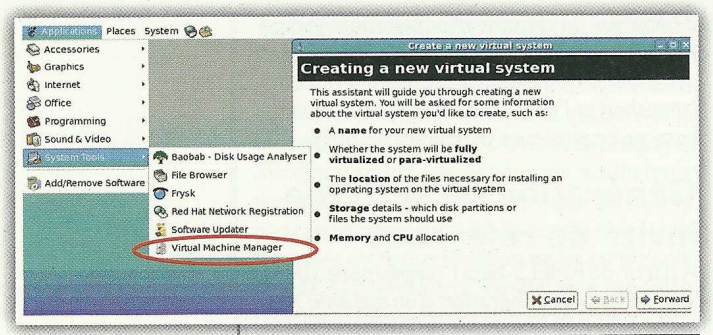


Figure 4 : Interface graphique de gestion des VM intégrée à Gnome et assistant de création

Ce gestionnaire graphique de machines virtuelles, spécifique à RedHat n'étant pas présent dans notre version encore en développement de RHEL5, nous avons dû le télécharger **[7]**, ainsi que ses nombreuses dépendances. Il est toutefois prévu de l'intégrer dans la version finale.

L'assistant graphique de création des machines virtuelles utilise, au final, le script interactif `xenguest-install` qui accepte aussi toutes les options nécessaires à la création d'une nouvelle VM. C'est d'ailleurs lui que nous allons utiliser pour créer notre premier système invité, grâce à la commande :

```
Xen# /usr/sbin/xenguest-install --name=vm --file=/dev/vg/vm \
--ram=512 --location=ftp://servername/kits/$DNAME --nographics
```

Cette commande créera une machine virtuelle nommée `vm`, possédant 512 MO de mémoire à la « mise sous tension ». Xen présentera le volume logique `/dev/vg/vm` au système invité afin qu'il puisse y placer son système d'exploitation en utilisant le protocole `ftp` à partir du serveur `servername`. Bien que dans notre cas, `servername` corresponde au domaine0 de notre serveur Xen, il est nécessaire de fournir le nom complet ou son adresse IP et, en aucun cas, `localhost`.

En effet, le paramètre `--location` sera transmis au système invité qui devra contacter via le réseau `servername` pour télécharger et installer le système d'exploitation. Comme indiqué plus haut, on aura pris soin de démarrer le service `ftp` sur `servername`. Si l'on souhaite utiliser le protocole `nfs` pour cette génération, la syntaxe est `--location='nfs:servername:/repertoire/de/la/distro'`.

L'envoi de la commande `xenguest-install` précédente génère le fichier de configuration `/etc/xen/vm`, démarre le domaine `vm` et présente à l'utilisateur la console `/dev/console`. Il s'ensuit une installation classique de RHEL5 au sein de ce domaine. Il est possible à tout moment de quitter cette console pour revenir à l'invite du domaine0 grâce à la séquence d'échappement `ctrl-]`, sans compromettre l'installation. Pour retourner à la console du domaine `vm`, il suffit d'envoyer la commande `xm console vm`.

Une fois la machine virtuelle générée et personnalisée à souhait, nous devons l'arrêter pour en faire plusieurs snapshots. Cet arrêt peut se faire dans le système invité (`init 0`) ou à partir du domaine0 grâce à la commande `xm shutdown vm`. Il est possible d'être plus violent avec `xm destroy vm`.

Création des copies du volume de référence

Les systèmes invités que nous allons créer n'utiliseront pas directement le volume logique du domaine `vm`, mais ses snapshots. De ce fait, `vm` ne sera pas modifié. Nous souhaitons quand même éviter tout risque de modification ou de corruption en le plaçant dans un mode de lecture seule :

```
Xen# lvchange -pr /dev/vg/vm
```

Notons que pour revenir à un mode lecture-écriture, il suffit d'envoyer la commande `lvchange -p rw /dev/vg/vm`.

En examinant la taille disponible sur notre groupe de volumes, nous constatons qu'il reste un peu plus de 8 GO :

```
Xen# vgdisplay /dev/vg | grep Free
  Free  PE / Size       2293 / 8.96 GB
```

Si nous affectons un espace de 1 GO par snapshot pour stocker les différences par rapport au volume de référence, nous pouvons donc créer 8 snapshots, que nous dénommerons `snap$i-vm` (`i` allant de 1 à 8). Ces snapshots constitueront les VBD des futurs domaines `vm1`, `vm2`, … `vm8`. La petite boucle suivante effectue la création des huit « photos » avec un tampon associé de 1 GO :

```
Xen# for i in 1 2 3 4 5 6 7 8 ; do
   lvcreate -s -L1024M -n snap$i-vm /dev/vg/vm
done
```

Les huit machines virtuelles devant être identiques, la création d'un fichier de configuration générique unique est possible. Nous placerons le numéro d'ordre de chaque machine dans la variable python « `vmid` » du fichier `/etc/xen/xmdefconfig`. La commande `xm` utilisée pour créer les domaines est prévue pour le consulter sans option supplémentaire. Ce fichier comporte une première partie de description des variables et de leurs valeurs limites, ainsi que des règles d'usage. Ensuite, on trouve la description proprement dite des machines virtuelles :

```
Xen# cat /etc/xen/xmdefconfig
#  -*- mode: python; -*-
def vmid_check(var, val):
    val = int(val)
    if val <= 0:
        raise ValueError
    if val > 8:
        raise ValueError
    return val

xm_vars.var('vmid',
         use="Virtual machine id. Integer greater than 0 and less than 9.",
         check=vmid_check)

# Check the defined variables have valid values..
xm_vars.check()
#----------------------------------------------------------------------------

disk = [ 'phy:/dev/vg/snap%d-vm,xvda,w' % vmid ]
vif = [ 'mac=00:16:3e:6b:54:e%d, bridge=xenbr0' % vmid ]
bootloader="/usr/bin/pygrub"
memory = "512"
name = "vm%d" % vmid
uuid = "12289a7e-b698-35bf-41a4-883efd880a4%d" % vmid
vcpus = 2
```

La variable `disk` décrit le VDB présenté à la VM (`/dev/vg/snap%d-vm`), le nom qu'il aura au sein de cette VM (`xvda`) et son mode

(`write`). Le `%d` sera remplacé par la variable `vmid` qui varie de 1 à 8. Ensuite, nous trouvons la variable `vif` décrivant une interface réseau virtuelle. Cette interface virtuelle, connectée au pont `xenbr0` permettra l'accès au réseau extérieur. La variable `bootloader` contient le chemin complet d'un script python mimant `grub` (figure 5) et qui propose à l'utilisateur la liste des noyaux utilisables au sein de notre VM.

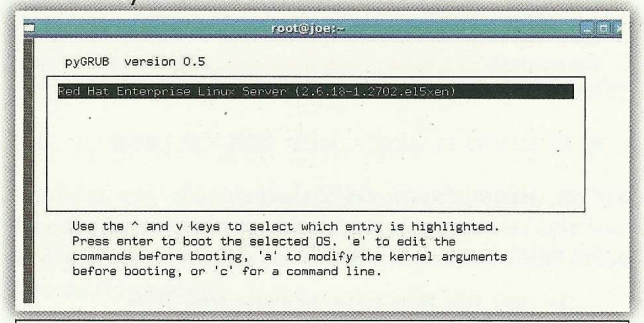


Figure 5 : Le bootloader pyGRUB

Pour créer un domaine et visualiser le démarrage de la VM associée, on enverra la commande :

```
Xen# xm create -c vmid=1
```

L'option `-c` de cette commande indique que l'on souhaite se connecter à la console du domaine. Pour la quitter, il faudra envoyer la séquence **`ctrl-]`**. Il est aussi possible de démarrer les huit VM simultanément grâce a la petite boucle :

```
for i in 1 2 3 4 5 6 7 8 ; do
xm create vmid=$I
done
```

La visualisation en temps réel de l'évolution de ces domaines en mode texte se fait avec `xentop`, ou alors en mode graphique grâce au `virt-manager` (figure 6).

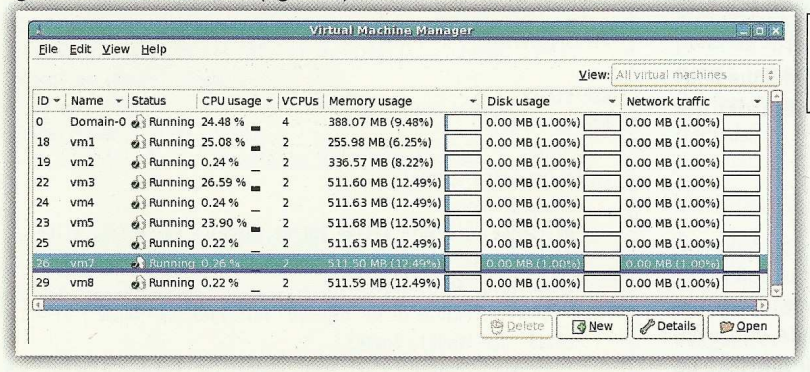


Figure 6 : Affichage principal du gestionnaire graphique de machines virtuelles

Remise en l'état

Lors de la création des snapshots, nous avons associé une zone de 1 GO pour stocker les copies en écriture de la VM, qui, encore une fois, ne peuvent pas se faire sur le volume logique original. Par exemple, si nous créons un fichier `/tmp/toto.txt`, il sera physiquement stocké dans cette zone de 1 GO.

Les commandes `lvdisplay` et `lvs` nous renseignent sur le taux de remplissage de cette zone, respectivement sous les dénominations « *allocated to snapshot* » et « *snap%* ». Il est fortement conseillé de surveiller de près ce taux de remplissage car, en cas de remplissage complet de cette zone, LVM2 place automatiquement le snapshot en lecture-seule, ce qui rend le système d'exploitation dans un état difficilement gérable.

Pour « purger » cette zone et donc se replacer dans l'état initial du système invité, nous allons arrêter le domaine, effacer le snapshot et le recréer. Toutes ces commandes pouvant être effectuées à partir du domaine0 :

```
Xen# xm shutdown vm8
Xen# lvremove -f /dev/vg/snap8-vm
Xen# lvcreate -s -L1024M -n snap8-vm /dev/vg/vm
Xen# xm create vmid=8
```

Ainsi, nous venons d'effectuer pas à pas une opération de type « undo » en utilisant une fonctionnalité du gestionnaire de volumes.

Conclusion

L'apparition d'une nouvelle technologie suscite toujours de nouveaux besoins. Dans le cas de la virtualisation, l'utilisateur final constatera rapidement des limitations au niveau du stockage. Son souhait le plus vif sera donc de mutualiser ce genre de ressource, au même titre que la puissance CPU, les entrées/sorties et la mémoire. Xen, associé aux solutions de Copie en Écriture récentes et présentes dans la plupart des distributions Linux, permet d'exaucer ce vœu. Une conséquence heureuse de cette mutualisation est la possibilité de disposer d'une manière simple pour replacer des machines virtuelles dans un état antérieur connu. Espérons que cette fonctionnalité fera le bonheur des équipes de test des logiciels du monde libre et même des autres…

François Donzé,

HP, Sophia-Antipolis,

francois.donze@hp.com

RÉFÉRENCES

- [1] http://www.cl.cam.ac.uk/Research/SRG/netos/xen/
- [2] http://fr.wikipedia.org/wiki/Copy-On-Write
- [3] http://www.xensource.com/
- [4] http://www.filesystems.org/project-unionfs.html
- [5] http://www.sistina.com/lvm/
- [6] http://evms.sourceforge.net
- [7] http://virt-manager.et.redhat.com/download.html

→ Manipulez avec les attributs

Laurent Gautrot

EN DEUX MOTS La perle de ce mois-ci a été rédigée par Laurent Gautrot, (l.gautrot@free.fr), de Paris.pm.

Contexte

Une personne m'a demandé s'il existe des moyens de sauvegarder les attributs des fichiers. Cette personne travaille dans le domaine du stockage et me faisait part d'une problématique de restauration particulière. Il lui arrive de restaurer des données et de réaliser que les attributs standards POSIX ne sont pas restaurés correctement. Il s'agit en réalité d'un sous-ensemble des attributs, et, en particulier, les modes, le couple propriétaire/groupe sous la forme UID/GID, puis les dates d'accès et de modification.

Une petite exploration sur le CPAN m'a orienté vers Chroniton, mais ce projet vise à fournir un système de sauvegarde complet, et la restauration des attributs sans les données semble hors de portée du système.

Comme ce travail semble tout indiqué pour Perl, voici une manière de sauvegarder des attributs.

La sauvegarde des attributs

L'utilisation de `Find::File` est incontournable, d'autant qu'il s'agit d'un module du noyau de Perl. Le but est de parcourir une liste de répertoires ou, à défaut, le répertoire courant et de lister toutes les sous-arborescences.

La fonction `wanted()`, appelée pour chaque fichier trouvé effectue un `lstat()` sur le fichier, puis affiche, sur la sortie standard, les informations souhaitées que sont le mode, les UID et GID, les dates d'accès et de modification et enfin du nom.

L'emploi de `lstat()` à la place de `stat()` est nécessaire pour ne pas déréférencer les liens symboliques. Enfin, et comme le mode renvoyé donne aussi le type du fichier, on lui applique un masque binaire avant de l'afficher sur la sortie standard en format CSV.

```
#!/usr/bin/perl

use strict;
use warnings;
use File::Find;

my @directories_to_search = (scalar @ARGV > 0) ? @ARGV : ('.');

find( \&wanted, @directories_to_search );

sub wanted {

    my ($dev,$ino,$mode,$nlink,$uid,$gid,$rdev,$size,
        $atime,$mtime,$ctime,$blksize,$blocks)
        = lstat($_);

    printf "%04o;%i;%i;%i;%i;%s\n",
       $mode & 07777, $uid, $gid, $atime, $mtime, $File::Find::
name;

  }
```

La restauration

Ici, le seul point délicat réside dans le placement du `chmod()` après le `chown()`. En effet, dans Perl, lors d'un changement de propriétaire, les *setuid bits* sont supprimés. Pour notre cas de figure, cela peut s'avérer très gênant.

```
#!/usr/bin/perl

use strict;
use warnings;

while (my $line = <>) {
    chomp $line;
    my ($mode,$uid,$gid,$atime,$mtime,$name) = split /;/, $line;
    print $name, ' ', $mode , ' ', oct($mode), "\n";
    chown $uid, $gid, $name
       or warn "Unable to set UID/GID '$uid/$gid' on '$name': $!\n";
    chmod (oct($mode), $name)
       or warn "Unable to set mode '$mode' on '$name': $!\n";
    utime $atime, $mtime, $name
       or warn "Unable to set atime/mtime '$atime/$mtime' on
'$name': $!\n";
  }
```

Un fichier de sauvegarde pourrait ressembler à ceci :

```
% perl attr2csv
0755;500;500;1160264381;1160264380;.
0644;500;500;1159903336;1159893644;./attr2.csv
0644;500;500;1160262923;1159903335;./csv2attr
0644;500;500;1160264387;1159339140;./attr2csv
```

En redirigeant la sortie standard vers un fichier, on dispose d'une sauvegarde des attributs. Ces informations peuvent être utilisées pour la restauration des attributs à l'aide de

la commande correspondante, mais peut aussi fournir une base d'information pour une vérification de changements d'attributs. C'est aussi bien en dessous de ce que peut proposer un outil comme AIDE.

On peut aussi imaginer une restauration des attributs d'une machine à une autre en utilisant une commande comme :

```
% perl attr2csv /emplacement/particulier | ssh root@autre_machine perl csv2attr
```

La réunion des amours

Comme ces charmants scripts sont simples, et liés fonctionnellement l'un à l'autre, il est possible de les fusionner et d'exécuter les parties adéquates en fonction du nom utilisé lors de l'invocation.

```
#!/usr/bin/perl

use strict;
use warnings;
use File::Find;


if ( $0 =~ m/attr2csv/ ) {
    my @directories_to_search = ( scalar @ARGV > 0 ) ? @ARGV : ('.');
    find( \&wanted, @directories_to_search );
}
elsif ( $0 =~ m/csv2attr/ ) {
   while ( my $line = <> ) {
      chomp $line;
      my ( $mode, $uid, $gid, $atime, $mtime, $name ) = split /;/, $line;
      print $name, ' ', $mode, ' ', oct($mode), "\n";
      chown $uid, $gid, $name
        or warn "Unable to set UID/GID '$uid/$gid' on '$name': $!\n";
      chmod( oct($mode), $name )
        or warn "Unable to set mode '$mode' on '$name': $!\n";
      utime $atime, $mtime, $name
        or warn "Unable to set atime/mtime '$atime/$mtime' on '$name':
$!\n";
    }
}
else {
    warn "$0 should be [attr2csv|csv2attr]\n";
}


sub wanted {
    my (
        $dev,  $ino,   $mode,  $nlink, $uid,     $gid,   $rdev,
        $size, $atime, $mtime, $ctime, $blksize, $blocks
    ) = lstat($_);
    printf "%04o;%i;%i;%i;%i;%s\n", $mode & 07777, $uid, $gid, $atime,
$ctime,
       $File::Find::name;
}
```

Le même script remplit les deux rôles. Il suffit de lier symboliquement le premier au second.

```
% ll attr2csv csv2attr
-rwxr-xr-x 1 lg lg 1073 oct 10 23:32 attr2csv
lrwxrwxrwx 1 lg lg    8 oct 10 23:32 csv2attr -> attr2csv
```

Et en uniligne ?

La notation est compacte et il est certain que l'on ne gagne pas énormément en lisibilité. Néanmoins, dans la partie de sauvegarde, le seul rôle de `File::Find` est de rechercher des fichiers, sans aucun critère. Au final, la commande `find(1)` remplit tout aussi bien ce rôle.

```
find /emplacement/particulier | perl -lane '($d,$i,$mode,$n,$uid,$gid,$r,$s,
    $atime,$mtime,$c,$bs,$b)=lstat($_); printf "%04o;%i;%i;%i;%i;%s\n",$mode &
    07777,$uid,$gid,$atime,$mtime,$_;'
```

La restauration part du même principe, on pourrait écrire un uniligne qui s'occupe de décortiquer les lignes pour réaliser cette opération.

```
perl -lane 'my ( $mode, $uid, $gid, $atime, $mtime, $name ) = split /;/;
    chown $uid, $gid, $name;chmod( oct($mode), $name ); utime $atime, $mtime,
    $name;' < fichier.csv
```

Je reconnais que pour un uniligne, il est très verbeux, avec trois `print()` de diagnostic.

Conclusion

En quelques lignes, il est possible d'écrire une procédure très simple et pourtant très puissante, qui s'applique aussi bien à de très petits répertoires qu'à des systèmes de fichiers avec quelques centaines de milliers d'inodes.

À vous !

Envoyez vos perles à perles@mongueurs.net. Elles seront peut-être publiées dans un prochain numéro de GNU/*Linux Magazine*.

Laurent Gautrot,

l.gautrot@free.fr

RÉFÉRENCES

- Chroniton : http://search.cpan.org/~jrockway/Chroniton/
- AIDE : http://www.cs.tut.fi/~rammer/aide.html

Dissection de Glib : l'analyseur de ligne de commande

Yves Mettier

EN DEUX MOTS La dissection de GLib nous amène ce mois-ci à étudier le code de l'analyseur de ligne de commande. Nous alternons un peu avec les structures de données vues dans les numéros précédents. GLib n'est pas qu'une boite d'outils dédiés aux structures de données !

Introduction

Historiquement, l'analyse de la ligne de commande était réalisée avec les fonctions du C, d'où une multitude de formats possibles. Certains utilisaient des tirets pour indiquer un argument, d'autres un slash, d'autres encore rien du tout. Souvent, pour ne pas dire presque toujours, les arguments devaient être indiqués dans l'ordre défini par le programme, ne laissant aucune souplesse à l'utilisateur.

Pour faciliter la tâche aux programmeurs, le jeu de fonctions dont `getopt()` est la plus connue est apparu. La façon d'indiquer les arguments s'est normée : les options étaient indiquées sous la forme d'un tiret suivi d'une lettre. Ceci n'était pas très causant et tout le monde n'adopta pas ce jeu de fonctions. GNU a tenté d'améliorer les choses avec `getopt_long()` qui permet d'utiliser les options longues, celles commençant par un double tiret. Cette fonction n'est cependant pas vraiment plus souple que la `getopt()` originale.

Par ailleurs, certains développèrent une bibliothèque dédiée à l'analyse de la ligne de commande, `popt`, utilisée entre autres par le logiciel de gestion de paquets `rpm`. L'analyseur de `Glib` se veut en fait un outil plus simple pouvant remplacer `popt`.

REMARQUE

Cet article est basé sur la version 2.12.4 de `Glib`.

Fonctionnalités

Jetons un coup d'œil à la documentation de `GLib` et de ce fameux analyseur de ligne de commande. Nous y trouvons trois structures de données : `GOptionContext`, `GOptionGroup` et `GOptionEntry`. Vu leurs noms, nous pouvons déjà deviner que nous allons décrire chacune de nos options dans des `GOptionEntry`, que nous allons grouper par `GOptionGroup`, et que nous allons insérer le tout dans un seul et unique `GOptionContext`.

Nous allons donc créer un jeu d'options principales et un autre jeu d'option secondaire :

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 #include <string.h>
04
05 #include <glib.h>
06
07 static gboolean o_debug=FALSE;
08 static gboolean o_verbose=FALSE;
09 static gint o_nb=1;
10 static gchar *o_chaine=NULL;
11 static gchar *o_fichier=NULL;
12
13 static GOptionEntry mes_options_principales[] =
14 {
15         { "debug", 'd', 0, G_OPTION_ARG_NONE, &o_debug, "Activer le
                                                    mode debug", NULL },
16         { "verbose", 'v', 0, G_OPTION_ARG_NONE, &o_verbose, "Activer
                                                  le mode blabla", NULL },
17         { "nb", 'n', 0, G_OPTION_ARG_INT, &o_nb, "Un nombre", "N" },
18         { "chaine", 0, 0, G_OPTION_ARG_STRING, &o_chaine, "Une chaîne
                                                  de caractères", "plouf" },
19         { NULL }
20 };
21
22 static GOptionEntry mes_options_secondaires[] =
23 {
24         { "fichier", 'f', 0, G_OPTION_ARG_FILENAME, &o_fichier, "Un
                                                  nom de fichier", NULL },
25         { NULL }
26 };
27
```

Il s'agira ensuite de les déclarer :

```
28 int main(int argc, char**argv) {
29
30         GError *error = NULL;
31         GOptionContext *context;
32         GOptionGroup *groupe_secondaire;
33
34 /* Création du contexte et ajout des entrées principales */
35         context = g_option_context_new ("Programme de test d'analyse
                                                  de ligne de commande");
36         g_option_context_add_main_entries (context, mes_options_
                                                  principales, NULL);
37
38 /* Création d'un groupe et ajout des entrées secondaires dans ce
                                                  groupe */
39         groupe_secondaire = g_option_group_new("fichiers", "Gestion
          des fichiers", "Options dédiées aux fichiers", NULL, NULL);
40         g_option_group_add_entries(groupe_secondaire, mes_options_
                                                  secondaires);
41
42 /* Ajout du groupe dans le contexte */
43         g_option_context_add_group (context, groupe_secondaire);
44
```

Analysez maintenant `argc` et `argv` :

```
45 /* Analyse de la ligne de commande */
```

```
46        g_option_context_parse (context, &argc, &argv, &error);
47
48 /* Affichage de choses et d'autres... */
49        printf("debug=%s verbose=%s nb=%d chaine='%s'\n",
o_debug?"TRUE":"FALSE",o_verbose?"TRUE":"FALSE", o_nb, o_chaine);
50        printf("fichier='%s'\n", o_fichier);
51
```

Un peu de nettoyage avant de finir ne fait jamais de mal !

```
52 /* Nettoyage */
53        g_option_context_free(context);
54
55 /* Au revoir, et merci pour le poisson ! */
56        exit(EXIT_SUCCESS);
57 }
```

Compilez et exécutez ce programme :

```
$ gcc `pkg-config --cflags --libs glib-2.0` test.c -o t
$ ./t
debug=FALSE verbose=FALSE nb=1 chaine='(null)'
fichier='(null)'
$ ./t --help Usage:
  t [OPTION...] Programme de test d'analyse de ligne de commande
Help Options:
  -?, --help            Show help options
  --help-all            Show all help options
  --help-fichiers       Options d?di?es aux fichiers
Application Options:
  -d, --debug           Activer le mode debug
  -v, --verbose         Activer le mode blabla
  -n, --nb=N            Un nombre
  --chaine=plouf        Une cha?ne de caract?res
$ ./t --help-all Usage:
  t [OPTION...] Programme de test d'analyse de ligne de commande
Help Options:
  -?, --help            Show help options
  --help-all            Show all help options
  --help-fichiers       Options d?di?es aux fichiers
Gestion des fichiers
  -f, --fichier         Un nom de fichier
Application Options:
  -d, --debug           Activer le mode debug
  -v, --verbose         Activer le mode blabla
  -n, --nb=N            Un nombre
  --chaine=plouf        Une cha?ne de caract?res
$ ./t --debug -n 3 --fichier='/bin/ls'
debug=TRUE verbose=FALSE nb=3 chaine='(null)'
fichier='/bin/ls'
```

Les structures

Attaquons les choses sérieuses. Que contiennent ces fameuses structures ? Ouvrez les fichiers `glib/goptions.c` et `glib/goption.h`.

GOptionContext

```
01 typedef struct _GOptionContext GOptionContext;
01 struct _GOptionContext
02 {
03   GList          *groups;
04
05   gchar          *parameter_string;
06   gchar          *summary;
07   gchar          *description;
08
09   GTranslateFunc   translate_func;
10   GDestroyNotify   translate_notify;
11   gpointer           translate_data;
12
13   guint            help_enabled   : 1;
14   guint            ignore_unknown : 1;
15
16   GOptionGroup    *main_group;
17
18   /* We keep a list of change so we can revert them */
19   GList          *changes;
20
21   /* We also keep track of all argv elements
22    * that should be NULLed or modified.
23    */
24   GList          *pending_nulls;
25 };
```

Nous ne pouvons pas encore comprendre les tenants et aboutissants de cette structure. Remarquez cependant que nous avons ligne 16 notre groupe principal et ligne 3 une liste de groupes supplémentaires. La suite est éclairée par la lecture de `g_option_context_new()` :

```
01 GOptionContext *
02 g_option_context_new (const gchar *parameter_string)
03
04 {
05   GOptionContext *context;
06
07   context = g_new0 (GOptionContext, 1);
08
09   context->parameter_string = g_strdup (parameter_string);
10   context->help_enabled = TRUE;
11   context->ignore_unknown = FALSE;
12
13   return context;
14 }
```

Cette fonction ne fait qu'allouer de la mémoire (ligne 7) et initialiser trois champs (lignes 9 à 11). Passons à la suite. Nous reviendrons à cette structure plus loin.

GOptionGroup

```
01 typedef struct _GOptionGroup   GOptionGroup;
01 struct _GOptionGroup
02 {
03   gchar          *name;
04   gchar          *description;
05   gchar          *help_description;
06
07   GDestroyNotify   destroy_notify;
08   gpointer         user_data;
09
10   GTranslateFunc   translate_func;
11   GDestroyNotify   translate_notify;
12   gpointer           translate_data;
13
14   GOptionEntry    *entries;
15   gint             n_entries;
16
17   GOptionParseFunc pre_parse_func;
18   GOptionParseFunc post_parse_func;
19   GOptionErrorFunc error_func;
20 };
```

Cette structure n'est pas beaucoup plus claire que la précédente. Remarquez cependant les premiers champs descriptifs (lignes 3 à 5) ainsi que l'emplacement où nous indiquerons nos entrées (lignes 14 et 15). Peut-être `g_option_group_new()` nous éclairera-t-il un peu plus ?

```
01 GOptionGroup *
02 g_option_group_new (const gchar    *name,
03                     const gchar    *description,
04                     const gchar    *help_description,
05                     gpointer        user_data,
06                     GDestroyNotify  destroy)
07
08 {
09   GOptionGroup *group;
10
11   group = g_new0 (GOptionGroup, 1);
12   group->name = g_strdup (name);
13   group->description = g_strdup (description);
14   group->help_description = g_strdup (help_description);
15   group->user_data = user_data;
16   group->destroy_notify = destroy;
17
18   return group;
19 }
```

Cela ne s'améliore pas : nous allouons de la mémoire pour le `GOptionGroup` et y insérons une copie des arguments.

GOptionEntry

La structure `GOptionEntry` nous en dira-t-elle plus ?

```
01 typedef struct _GOptionEntry   GOptionEntry;
01 struct _GOptionEntry
02 {
03   const gchar *long_name;
04   gchar        short_name;
05   gint         flags;
06
07   GOptionArg   arg;
08   gpointer     arg_data;
09
10   const gchar *description;
11   const gchar *arg_description;
12 };
```

Voilà, nous y sommes. Pour chaque argument, c'est ici que nous stockons les informations qui lui sont associées. Le `GOptionArg` est juste une énumération :

```
01 typedef enum
02 {
03   G_OPTION_ARG_NONE,
04   G_OPTION_ARG_STRING,
05   G_OPTION_ARG_INT,
06   G_OPTION_ARG_CALLBACK,
07   G_OPTION_ARG_FILENAME,
08   G_OPTION_ARG_STRING_ARRAY,
09   G_OPTION_ARG_FILENAME_ARRAY,
10   G_OPTION_ARG_DOUBLE,
11   G_OPTION_ARG_INT64
12 } GOptionArg;
```

Il indique vraisemblablement le type de l'argument attendu.

La création du contexte

g_option_context_add_main_entries()

Dans notre programme du début, après avoir utilisé `g_option_context_new()`, nous avons fait appel à `g_option_context_add_main_entries()` :

```
01 void
02 g_option_context_add_main_entries (GOptionContext      *context,
03                                    const GOptionEntry  *entries,
04                                    const gchar         *translation_domain)
05 {
06   g_return_if_fail (entries != NULL);
07
08   if (!context->main_group)
09     context->main_group = g_option_group_new (NULL, NULL, NULL, NULL, NULL);
10
11   g_option_group_add_entries (context->main_group, entries);
12   g_option_group_set_translation_domain (context->main_group,
                                            translation_domain);
13 }
```

Cette fonction n'est finalement qu'une fonction d'appoint pour créer le groupe principal s'il n'existe pas déjà (lignes 8 et 9) et appeler `g_option_group_add_entries()` et `g_option_group_set_translation_domain()` (lignes 11 et 12). C'est exactement ce que nous avons fait pour notre groupe d'options secondaire à ceci près que l'internationalisation est prise en compte avec le domaine de traduction (ligne 12).

g_option_group_add_entries()

Ce qui nous intéresse donc, finalement, c'est `g_option_group_add_entries()` :

```
01 g_option_group_add_entries (GOptionGroup       *group,
02                             const GOptionEntry *entries)
03 {
04   gint i, n_entries;
05
06   g_return_if_fail (entries != NULL);
07
08   for (n_entries = 0; entries[n_entries].long_name != NULL; n_entries++) ;
09
10   group->entries = g_renew (GOptionEntry, group->entries,
                               group->n_entries + n_entries);
11
12   memcpy (group->entries + group->n_entries, entries,
                 sizeof (GOptionEntry) * n_entries);
13
14   for (i = group->n_entries; i < group->n_entries + n_entries; i++)
15     {
16       gchar c = group->entries[i].short_name;
17
18       if (c)
19         {
20           if (c == '-' || !g_ascii_isprint (c))
21             {
22               g_warning (G_STRLOC": ignoring invalid short option '%c' (%d)",
                                                                    c, c);
23               group->entries[i].short_name = 0;
24             }
25         }
26     }
27
28   group->n_entries += n_entries;
29 }
```

Ligne 8, cette boucle sans contenu est un classique pour compter le nombre d'éléments d'une structure, ici de `entries`. La variable `n_entries` y est incrémentée jusqu'au dernier élément.

Ligne 10, la zone des entrées est agrandie de `n_entries` fois la taille d'un `GOptionEntry`. Le champ `entries` de cette structure correspond donc à un tableau. Vous devinez ligne 28 que la taille de ce tableau est `group->n_entries` et c'est pourquoi il est incrémenté de `n_entries` à cette ligne.

Ligne 12, le tableau des entrées indiqué en argument est concaténé aux entrées existantes grâce à un `memcpy()`.

Cela devrait suffire, mais lignes 14 à 26, nous avons droit à une boucle de vérification. En effet, pour chaque entrée, l'option courte est testée. Elle ne doit pas valoir `'-'`, ce qui signifierait `--` et correspondrait à la fin des options. Elle doit par contre valoir un caractère faisant partie des caractères imprimables (fonction `g_ascii_isprint()`). Si ce test ligne 20 était positif, un message d'avertissement serait envoyé au développeur (ligne 22). Celui-ci devrait corriger l'option afin que ce message n'apparaisse pas à l'utilisateur final. L'option est de plus désactivée ligne 23.

g_option_group_set_translation_domain()

Intéressons-nous à cette fonction :

```
01 void
02 g_option_group_set_translation_domain (GOptionGroup *group,
03                                       const gchar  *domain)
04 {
05   g_return_if_fail (group != NULL);
06
07   g_option_group_set_translate_func (group,
08                      (GTranslateFunc)dgettext_swapped,
09                      g_strdup (domain),
10                      g_free);
11 }
```

Nous avons droit à nouveau, comme dans les articles précédents, à un jeu de piste !

```
01 void
02 g_option_group_set_translate_func (GOptionGroup   *group,
03                      GTranslateFunc  func,
04                      gpointer        data,
05                      GDestroyNotify  destroy_notify)
06 {
07   g_return_if_fail (group != NULL);
08
09   if (group->translate_notify)
10     group->translate_notify (group->translate_data);
11
12   group->translate_func = func;
13   group->translate_data = data;
14   group->translate_notify = destroy_notify;
15 }
```

Cette fonction se contente de changer les fonctions et données de traduction. Si une fonction précédente avait été indiquée pour notifier quelqu'un du changement de fonction dans `group->translate_notify`, elle est appelée ligne 9. La nouvelle est indiquée ligne 14 après avoir changé les champs nécessaires lignes 12 et 13 pour la nouvelle fonction et la donnée utilisateur.

g_option_context_add_group()

Nous avons vu comment créer un groupe et, dans le cas du groupe principal, l'intégrer dans le contexte. Il ne nous reste plus qu'à étudier comment prendre également en compte des groupes secondaires. C'est le rôle de `g_option_context_add_group()` :

```
01 void
02 g_option_context_add_group (GOptionContext *context,
03                             GOptionGroup   *group)
04 {
05   GList *list;
06
07   g_return_if_fail (context != NULL);
08   g_return_if_fail (group != NULL);
09   g_return_if_fail (group->name != NULL);
10   g_return_if_fail (group->description != NULL);
11   g_return_if_fail (group->help_description != NULL);
12
13   for (list = context->groups; list; list = list->next)
14     {
15       GOptionGroup *g = (GOptionGroup *)list->data;
16
17       if ((group->name == NULL && g->name == NULL) ||
18           (group->name && g->name && strcmp (
                  group->name, g->name) == 0))
19         g_warning ("A group named \"%s\" is already"
                       "part of this GOptionContext",
20                    group->name);
21     }
22
23   context->groups = g_list_append (context->groups, group);
24 }
```

Les groupes sont stockés dans le contexte en tant que liste chaînée. Nous l'avions déjà supposé en voyant le type du champ `groups` dans un `GOptionContext`. Nous en avons la preuve ici : notre groupe est inséré aux autres avec `g_list_append()` ligne 23. Le code précédent, lignes 13 à 21 est une petite vérification : tous les groupes existants sont parcourus (boucle ligne 13) et leur nom est comparé à celui du groupe que nous voulons ajouter (test lignes 17 et 18). Si tel était le cas, un petit message d'avertissement est produit lignes 19 et 20.

L'analyse de la ligne de commande

Nous voici arrivés à ce qui nous intéresse le plus : l'analyse de la ligne de commande. Malheureusement, les lignes de code sont trop nombreuses pour être reproduites ici. Nous allons donc devoir nous limiter à certaines parties. Nous ne montrerons donc en particulier pas les parties répétitives, ni celles ne présentant pas grand intérêt comme les tests des arguments ou les initialisations basiques.

```
01 gboolean
02 g_option_context_parse (GOptionContext   *context,
03                         gint             *argc,
04                         gchar          ***argv,
05                         GError         **error)
06 {
```

Ce qui suit détermine le nom du programme et fait appel à `g_set_prgname()`. Puis une boucle parcourt la liste des groupes (`context-`

>groups) et y exécute, pour chacun, la fonction pre_parse_func() qui lui est associée (par le champ du même nom) si celle-ci est définie. Cette fonction est également exécutée pour le groupe principal context->main_group si elle est définie. Un code similaire se trouve plus loin pour la gestion des erreurs. Nous arrivons à la ligne 48 qui démarre l'analyse des arguments.

```
48  if (argc && argv)
49    {
50      gboolean stop_parsing = FALSE;
51      gboolean has_unknown = FALSE;
52      gint separator_pos = 0;
53
54      for (i = 1; i < *argc; i++)
55        {
56          gchar *arg, *dash;
57          gboolean parsed = FALSE;
58
59          if ((*argv)[i][0] ==
          '-' && (*argv)[i][1] != '\0' && !stop_parsing)
60            {
61              if ((*argv)[i][1] == '-')
62                {
63                  /* -- option */
64
```

Pour chaque argument, nous testons s'il commence par un tiret (boucle ligne 54 et test ligne 59). S'il n'y a pas de problème, nous avons deux cas à distinguer : les options courtes et les options longues. Nous allons voir les options longues et faire l'impasse sur les options courtes, le code étant relativement le même. En cas de problème, ce n'est pas vraiment un problème. Nous avons seulement affaire à un argument non référencé qui sera traité lignes 234 à 240.

```
65                  arg = (*argv)[i] + 2;
66
67                  /* '--' terminates list of arguments */
68                  if (*arg == 0)
69                    {
70                      separator_pos = i;
71                      stop_parsing = TRUE;
72                      continue;
73                    }
74
```

Ligne 65, nous faisons pointer arg sur le nom de l'option. Si celui-ci est nul, l'option était -- qui signifie que nous devons arrêter l'analyse. C'est l'objet de la ligne 71.

```
75                  /* Handle help options */
76                  if (context->help_enabled)
77                    {
78                      if (strcmp (arg, "help") == 0)
79                        print_help (context, TRUE, NULL);
80                      else if (strcmp (arg, "help-all") == 0)
81                        print_help (context, FALSE, NULL);
82                      else if (strncmp (arg, "help-", 5) == 0)
83                        {
84                          GList *list;
85
86                          list = context->groups;
87
```

```
88                          while (list)
89                            {
90                              GOptionGroup *group = list->data;
91
92                              if (strcmp (arg + 5, group->name) == 0)
93                                print_help (context, FALSE, group);
94
95                              list = list->next;
96                            }
97                        }
98                    }
99
```

Il est possible de désactiver l'aide en mettant FALSE dans context->help_enabled. C'est d'ailleurs l'objet de la fonction g_option_context_set_help_enabled() dont nous ne parlerons pas plus ici. Si elle est activée (test ligne 76), nous devons tester si l'option est un dérivé de help. Vous faites ici connaissance avec la fonction print_help() dont nous avons toutes les utilisations possibles lignes 79, 81 et 93. Son premier argument indique le contexte. Le deuxième indique si nous montrons l'aide du groupe principal. Le dernier correspond à un groupe dont il faut afficher l'aide. Vous verrez plus loin que la fonction print_help() se termine par un appel à exit().

```
100                 if (context->main_group &&
101                     !parse_long_option (context,
                        context->main_group, &i, arg,
102                           FALSE, argc, argv, error, &parsed))
103                   goto fail;
104
105                 if (parsed)
106                   continue;
107
108                 /* Try the groups */
109                 list = context->groups;
110                 while (list)
111                   {
112                     GOptionGroup *group = list->data;
113
114                     if (!parse_long_option (context, group, &i, arg,
115                           FALSE, argc, argv, error, &parsed))
116                       goto fail;
117
118                     if (parsed)
119                       break;
120
121                     list = list->next;
122                   }
123
124                 if (parsed)
125                   continue;
126
```

Pour chaque groupe, et tant que nous n'avons pas trouvé l'entrée correspondante, nous cherchons si elle y est avec parse_long_option(). Nous commençons avec le groupe principal ligne 101. Si elle y est (test ligne 105), nous passons à la suite (ligne 106). Sinon, nous attaquons la liste des groupes secondaires (boucle ligne 110) dont nous sortons ligne 119 si nous avons trouvé. Lignes 124 et 125, nous passons à la suite si l'option était dans un groupe.

```
127                 /* Now look for --<group>-<option> */
128                 dash = strchr (arg, '-');
129                 if (dash)
130                   {
```

```
131             /* Try the groups */
132             list = context->groups;
133             while (list)
134               {
135                 GOptionGroup *group = list->data;
136
137                 if (strncmp (group->name, arg,
                                 dash - arg) == 0)
138                   {
139                     if (!parse_long_option (context,
                            group, &i, dash + 1,
140                            TRUE, argc, argv, error, &parsed))
141                       goto fail;
142
143                     if (parsed)
144                       break;
145                   }
146
147                 list = list->next;
148               }
149           }
150
```

Vous découvrez ici qu'il est également possible de faire précéder le nom de l'option par le nom du groupe correspondant, avec un tiret séparateur. Si ce tiret existe (test ligne 128), nous recherchons le groupe portant le nom correspondant (boucle ligne 133 et test ligne 137). Pour ce groupe, nous faisons appel à la fonction `parse_long_option()` (ligne 139) comme précédemment.

```
151               if (context->ignore_unknown)
152                 continue;
153             }
154           else
155             { /* short option */
```

Si les options inconnues sont ignorées, nous itérons de force (ligne 152). Sinon, nous trouverons du code dédié aux options inconnues plus loin. Nous passons aux options courtes, mais ce code ressemblant au précédent, nous allons utiliser le sécateur jusqu'à la ligne 217. Vous avez ci-dessous le code correspondant à une option non analysée. Si `context->ignore_unknown` est nul, nous générons une erreur et, comble de l'horreur, exécutons un `goto` !

```
218             }
219
220           if (!parsed)
221             has_unknown = TRUE;
222
223           if (!parsed && !context->ignore_unknown)
224             {
225               g_set_error (error,
226                            G_OPTION_ERROR,
                             G_OPTION_ERROR_UNKNOWN_OPTION,
227                                _("Unknown option %s"),
                                          (*argv)[i]);
228               goto fail;
229             }
```

Ce `goto` est utilisé à bon escient. Les langages plus évolués que le C proposent un mécanisme qui, en soulevant une erreur, nous envoie directement et de façon inconditionnelle à une portion de code dédiée à la gestion des erreurs. C'est le cas avec `try` et `catch` en java par exemple. Ici, la gestion de l'erreur s'effectue au label `fail`.

```
230         }
```

```
231       else
232         {
233           /* Collect remaining args */
234           if (context->main_group &&
235               !parse_remaining_arg (context, context->main_group, &i,
236                                     argc, argv, error, &parsed))
237             goto fail;
238
239           if (!parsed && (has_unknown || (*argv)[i][0] == '-'))
240             separator_pos = 0;
241         }
242     }
```

Nous avons ci-dessus la gestion des arguments au-delà de l'option -- ou des arguments non précédés par un tiret. Ils correspondent à la condition négative du test ligne 59.

Nous avons ci-dessous, où plutôt nous avions avant que l'auteur n'efface les lignes, du code pour exécuter les fonctions de post-analyse, de la même façon que nous avions au début les fonctions de pré-analyse de la ligne de commande. Elles ne présentent pas d'intérêt. Vous trouverez un code similaire plus loin pour la gestion des erreurs. Nous allons directement à la fin où les arguments sont une nouvelle fois analysés afin de supprimer ceux qui ont été traités. Ne restent donc que ceux dont l'analyseur n'a su que faire. Par ailleurs, si `context->ignore_unknown` est vrai, ce code n'a pas lieu d'être exécuté.

```
272   if (argc && argv)
273     {
274       free_pending_nulls (context, TRUE);
275
276       for (i = 1; i < *argc; i++)
277         {
278           for (k = i; k < *argc; k++)
279             if ((*argv)[k] != NULL)
280               break;
281
282           if (k > i)
283             {
284               k -= i;
285               for (j = i + k; j < *argc; j++)
286                 {
287                   (*argv)[j-k] = (*argv)[j];
288                   (*argv)[j] = NULL;
289                 }
290               *argc -= k;
291             }
292         }
293     }
294
295   return TRUE;
296
```

Nous arrivons enfin à la gestion des erreurs et au label `fail`. Ce code est similaire à ceux qui font appel aux fonctions de pré-analyse au début et aux fonctions de post-analyse au milieu de la fonction `g_option_context_parse()`. Au lieu d'utiliser les champs `pre_parse_func()` ou `post_parse_func()`, il s'agit ici de `error_func()`. Pour chaque groupe (boucle ligne 301), nous testons si une telle fonction existe (ligne 305)

et, le cas échéant, nous l'exécutons (ligne 306). Cette démarche est également effectuée pour le groupe principal (test ligne 312 et exécution ligne 313).

```
297  fail:
298
299   /* Call error hooks */
300   list = context->groups;
301   while (list)
302     {
303       GOptionGroup *group = list->data;
304
305       if (group->error_func)
306         (* group->error_func) (context, group,
307                                group->user_data, error);
308
309       list = list->next;
310     }
311
312   if (context->main_group && context->main_group->error_func)
313     (* context->main_group->error_func) (context,
                                 context->main_group,
314        context->main_group->user_data, error);
315
316   free_changes_list (context, TRUE);
317   free_pending_nulls (context, FALSE);
318
319   return FALSE;
320 }
```

Un peu de nettoyage lignes 316 et 317 et nous voici avec quelques fonctions qui nous restent sur les bras.

parse_long_option()

Commençons avec cette fonction. Les curieux iront également voir `parse_short_option()` dont le code est proche de celle-ci. Juste avant, voyons une macro, `NO_ARG` dont nous allons avoir besoin très vite :

```
01 #define NO_ARG(entry) ((entry)->arg == G_OPTION_ARG_NONE ||       \
02                        ((entry)->arg == G_OPTION_ARG_CALLBACK &&  \
03                         ((entry)->flags & G_OPTION_FLAG_NO_ARG)))
```

Cette macro teste si l'option est censée prendre un argument ou non. Nous la retrouvons dans le code suivant, ligne 22.

Cette fonction n'est rien d'autre qu'une boucle (ligne 14) qui recherche la bonne option, ou plutôt l'entrée du groupe dont le nom (long, puisque nous ne nous occupons ici que des options longues) de l'option correspond à celle en cours d'analyse.

```
01 static gboolean
02 parse_long_option (GOptionContext *context,
03                    GOptionGroup   *group,
04                    gint           *index,
05                    gchar          *arg,
06                    gboolean        aliased,
07                    gint           *argc,
08                    gchar        ***argv,
09                    GError       **error,
10                    gboolean       *parsed)
11 {
12   gint j;
13
14   for (j = 0; j < group->n_entries; j++)
15   {
16     if (*index >= *argc)
17       return TRUE;
18
19     if (aliased && (group->entries[j].flags & G_OPTION_FLAG_NOALIAS))
20       continue;
21
```

Nous commençons par tester le cas d'un simple drapeau, soit d'une option au format `--option` sans argument supplémentaire. Le test suivant vérifie l'égalité entre l'entrée courante et l'option courante ainsi que le fait que l'option ne prenne pas d'argument. Si tel est le cas, nous exécutons `parse_arg()` dont nous verrons le code plus loin.

```
22       if (NO_ARG (&group->entries[j]) &&
23           strcmp (arg, group->entries[j].long_name) == 0)
24         {
25           gchar *option_name;
26
27           option_name = g_strconcat ("--",
             group->entries[j].long_name, NULL);
28           parse_arg (context, group, &group->entries[j],
29                      NULL, option_name, error);
30           g_free(option_name);
31
32           add_pending_null (context, &((*argv)[*index]), NULL);
33           *parsed = TRUE;
34         }
35       else
36         {
```

Nous n'avons pas affaire à une simple option, mais à une qui prend un argument. Nous testons si celui-ci a été indiqué sous la forme `--option=argument` ou `--option argument` ligne 39. Sinon, l'option n'est pas analysée (fin du bloc d'exécution ligne 105).

```
37     gint len = strlen (group->entries[j].long_name);
38
39     if (strncmp (arg, group->entries[j].long_name, len) == 0 &&
40         (arg[len] == '=' || arg[len] == 0))
41       {
42         gchar *value = NULL;
43         gchar *option_name;
44
45         add_pending_null (context, &((*argv)[*index]), NULL);
46         option_name = g_strconcat ("--",
           group->entries[j].long_name, NULL);
47
```

Ci-dessus, nous avons obtenu le nom de l'option. Ci-après, nous allons chercher à obtenir sa valeur. Le cas `--option=argument` est simple et traité en deux lignes (48 et 49). Nous entrons ensuite dans divers autres cas que nous n'allons pas détailler :

- ligne 50 : nous ne sommes pas encore au dernier argument. Nous traitons le cas où l'argument est facultatif (test ligne 52). Si ce n'est pas le cas, nous avons la valeur ligne 54. S'il est effectivement facultatif, il faut vérifier l'argument suivant : commence-t-il par un tiret (test ligne 60) ? Si oui, nous analysons l'option directement ligne 63 et quittons ligne 67. Sinon, nous avons affaire à la valeur qui a été donnée sur la ligne de commande ;
- ligne 77 : nous sommes au dernier argument, mais pour cette option, la valeur est facultative (nous sommes donc dans le cas où cette valeur n'a pas été indiquée). La fonction `parse_arg()` est appelée avec `NULL` en guise de quatrième argument, la valeur ligne 67. Sinon, nous avons affaire à la valeur qui a été donnée sur la ligne de commande. Ce code est identique aux lignes 62 à 67 ;
- ligne 87 : tous les autres cas sont des cas d'erreur : il manque un argument. La fonction retourne `FALSE` après avoir appelé `g_set_error()`.

```
48              if (arg[len] == '=')
49                value = arg + len + 1;
50              else if (*index < *argc - 1)
51                {
52                  if (!(group->entries[j].flags
                    & G_OPTION_FLAG_OPTIONAL_ARG))
53                    {
54                      value = (*argv)[*index + 1];
55                      add_pending_null (context,
                                &((*argv)[*index + 1]), NULL);
56                      (*index)++;
57                    }
58                  else
59                    {
60                      if ((*argv)[*index + 1][0] == '-')
61                        {
62                          gboolean retval;
63                          retval = parse_arg (context,
                              group, &group->entries[j],
64                                    NULL, option_name, error);
65                          *parsed = TRUE;
66                          g_free (option_name);
67                            return retval;
68                        }
69                      else
70                        {
71                          value = (*argv)[*index + 1];
72                          add_pending_null (context,
                            &((*argv)[*index + 1]), NULL);
73                          (*index)++;
74                        }
75                    }
76                }
77              else if (*index >= *argc - 1 &&
78                       group->entries[j].flags &
                         G_OPTION_FLAG_OPTIONAL_ARG)
79                {
[ code identique aux lignes 62 à 67 ]
86                }
87              else
[ code de gestion d'erreur ]
```

Lorsque nous arrivons ici, nous avons une option nommée `option_name` et sa valeur `value`. Analysons cela avec `parse_arg()` avant de faire le ménage et de retourner d'où nous venons :

```
96            if (!parse_arg (context, group, &group->entries[j],
97                            value, option_name, error))
98              {
99                g_free (option_name);
100                return FALSE;
101              }
102
103            g_free (option_name);
104            *parsed = TRUE;
105          }
106      }
107  }
108
109  return TRUE;
110 }
```

parse_remaining_arg()

Lorsque nous sommes au-delà de l'option -- ou que nous avons un argument non précédé par un tiret, nous avons vu plus haut que nous devions appeler `parse_remaining_arg()` :

```
static gboolean
parse_remaining_arg (GOptionContext *context,
                     GOptionGroup   *group,
                     gint           *index,
                     gint           *argc,
                     gchar        ***argv,
                     GError        **error,
                     gboolean       *parsed)
{
  gint j;

  for (j = 0; j < group->n_entries; j++)
    {
      if (*index >= *argc)
                    return TRUE;

      if (group->entries[j].long_name[0])
                    continue;

      g_return_val_if_fail (group->entries[j].arg ==
G_OPTION_ARG_STRING_ARRAY ||
group->entries[j].arg == G_OPTION_ARG_FILENAME_ARRAY, FALSE);

      add_pending_null (context, &((*argv)[*index]), NULL);

      if (!parse_arg (context, group, &group->entries[j],
(*argv)[*index], "", error))
                    return FALSE;

      *parsed = TRUE;
      return TRUE;
    }

  return TRUE;
}
```

parse_arg()

Nous avons vu des appels à cette fonction qui, comme son nom l'indique, analyse un argument. En voici le code :

```
01 static gboolean
02 parse_arg (GOptionContext *context,
03            GOptionGroup   *group,
04            GOptionEntry   *entry,
05            const gchar    *value,
06            const gchar    *option_name,
07            GError        **error)
08
09 {
10   Change *change;
11
12   g_assert (value || OPTIONAL_ARG (entry) || NO_ARG (entry));
13
14   switch (entry->arg)
15     {
16     case G_OPTION_ARG_NONE:
17       {
18         change = get_change (context, G_OPTION_ARG_NONE,
19                              entry->arg_data);
20
21         *(gboolean *)entry->arg_data = !(entry->flags &
                                    G_OPTION_FLAG_REVERSE);
22          break;
23       }
24     case G_OPTION_ARG_STRING:
25       {
26         gchar *data;
27
28         data = g_locale_to_utf8 (value, -1, NULL, NULL, error);
29
```

```
30        if (!data)
31          return FALSE;
32
33        change = get_change (context, G_OPTION_ARG_STRING,
34                             entry->arg_data);
35        g_free (change->allocated.str);
36
37        change->prev.str = *(gchar **)entry->arg_data;
38        change->allocated.str = data;
39
40        *(gchar **)entry->arg_data = data;
41        break;
42      }
43    case G_OPTION_ARG_STRING_ARRAY:
[...]
75    case G_OPTION_ARG_FILENAME:
[...]
98    case G_OPTION_ARG_FILENAME_ARRAY:
[...]
133    case G_OPTION_ARG_INT:
134      {
135        gint data;
136
137        if (!parse_int (option_name, value,
138                        &data,
139                        error))
140          return FALSE;
141
142        change = get_change (context, G_OPTION_ARG_INT,
143                             entry->arg_data);
144        change->prev.integer = *(gint *)entry->arg_data;
145        *(gint *)entry->arg_data = data;
146        break;
147      }
148    case G_OPTION_ARG_CALLBACK:
[...]
180    case G_OPTION_ARG_DOUBLE:
[...]
197    case G_OPTION_ARG_INT64:
[...]
214    default:
215      g_assert_not_reached ();
216    }
217
218  return TRUE;
219 }
```

Le code se répétant plus ou moins, nous avons éliminé le code correspondant à la plupart des cas. Étudions néanmoins le cas d'une chaîne de caractères, puis celui d'un entier. Nous avons d'abord une pré-analyse via une transformation en UTF8 (ligne 28) pour une chaîne, alors que pour un entier, nous appelons `parse_int()` (ligne 137). Puis la fonction `get_change()` se charge de créer le nécessaire pour sauvegarder la valeur initiale de l'argument. La sauvegarde a lieu lignes 37 et 38 pour une chaîne, et ligne 144 pour un entier. Puis nous modifions la ligne `entry` correspondant à notre argument dans le tableau des options, lignes 40 ou 145.

Normalement, le cas par défaut ne devrait jamais survenir. Aussi, plutôt que de ne rien mettre, les développeurs préfèrent en général provoquer une erreur si cela arrivait, ce qui est le cas ligne 215.

Voyons encore la fonction `get_change()` dont nous avons parlé précédemment :

```
01 static Change *
02 get_change (GOptionContext *context,
03             GOptionArg      arg_type,
04             gpointer        arg_data)
05 {
06   GList *list;
07   Change *change = NULL;
08
09   for (list = context->changes; list != NULL; list = list->next)
10     {
11       change = list->data;
12
13       if (change->arg_data == arg_data)
14         goto found;
15     }
16
17   change = g_new0 (Change, 1);
18   change->arg_type = arg_type;
19   change->arg_data = arg_data;
20
21   context->changes = g_list_prepend (context->changes, change);
22
23 found:
24
25   return change;
26 }
```

Cette fonction recherche tout simplement si une sauvegarde a déjà eu lieu. Le cas échant, vous voyez alors un `goto` ligne 14 qui n'est pas moins propre que le `return change` que vous auriez pu lire à la place. Sinon, un nouveau *changement* est créé ligne 17 et pré-rempli lignes 18 et 19. Ce changement est ajouté au début de la liste des changements ligne 21.

L'intérêt de sauvegarder les valeurs précédentes est de pouvoir les restaurer. Si vous faites une recherche sur « changes » dans `glib/goption.c`, c'est dans `get_change()` que vous trouverez le plus d'occurrences évidemment, mais vous en trouvez également dans `free_changes_list()`. Le second argument de cette fonction est explicite : il s'appelle `revert`. Il s'agit donc bien d'un retour en arrière. Vous pouvez maintenant mieux comprendre la fin du code de `g_option_context_parse()` que nous avons vu précédemment : lors d'un échec, nous avions vu un appel à `goto fail`. A la suite du label `fail`, vous trouvez ligne 316 l'appel à `free_changes_list()`.

print_help()

Enfin, nous arrivons à la fonction `print_help()` qui affiche l'aide. Mais avant de lire son code, voyez la macro `TRANSLATE` qui est utilisée plusieurs fois dans `print_help()` :

```
01 #define TRANSLATE(group, str) (((group)->translate_func ? (* (group)->translate_func) ((str), (group)->translate_data) : (str)))
```

S'il existe une fonction de traduction définie pour le groupe indiqué, elle est utilisée pour traduire `str`. Sinon, c'est `str` elle-même qui est utilisée.

Voici maintenant la tant attendue `print_help()`. Ce code est assez long, mais comme peu de choses sont redondantes, nous allons le voir en entier :

```
01 static void
02 print_help (GOptionContext *context,
03             gboolean        main_help,
04             GOptionGroup   *group)
05 {
```

```
06  GList *list;
07  gint max_length, len;
08  gint i;
09  GOptionEntry *entry;
10  GHashTable *shadow_map;
11  gboolean seen[256];
12  const gchar *rest_description;
13
```

Les options restantes et le résumé

Le code suivant analyse les options restantes. Nous découvrons cette fonctionnalité en lisant le code, et c'est en recherchant `arg_description` dans `glib/goption.c` puis `G_OPTION_REMAINING` dans ce même fichier et ensuite dans la documentation officielle que nous avons l'explication. Si dans vos entrées vous en définissez une de type `G_OPTION_ARG_STRING_ARRAY` ou `G_OPTION_ARG_FILENAME_ARRAY`, et que vous mettez `G_OPTION_REMAINING` en guise d'option longue (la chaîne correspondante est une chaîne vide : `#define G_OPTION_REMAINING ""`), alors sur la ligne `votre_programme [OPTION...]`... vous trouverez la description de ces options restantes. L'affichage de cette ligne a lieu lignes 29 à 34 et la recherche de la bonne description se trouve dans la boucle lignes 18 à 26.

Le résumé est affiché ligne 37 s'il existe (test ligne précédente).

```
14  rest_description = NULL;
15  if (context->main_group)
16    {
17
18      for (i = 0; i < context->main_group->n_entries; i++)
19        {
20          entry = &context->main_group->entries[i];
21          if (entry->long_name[0] == 0)
22            {
23              rest_description =
        TRANSLATE (context->main_group, entry->arg_description);
24              break;
25            }
26        }
27    }
28
29  g_print ("%s\n  %s %s%s%s%s%s\n\n",
30           _("Usage:"), g_get_prgname(), _("[OPTION...]"),
31           rest_description ? " " : "",
32           rest_description ? rest_description : "",
33           context->parameter_string ? " " : "",
34           context->parameter_string ? TRANSLATE (context,
                        context->parameter_string) : "");
35
36  if (context->summary)
37    g_print ("%s\n\n", TRANSLATE (context, context->summary));
38
```

Résolution des conflits

Nous allons maintenant résoudre les conflits, à savoir les options portant le même nom long et celles le même nom court. Pour cela, nous effectuons une première passe lignes 42 à 56. Pour les options longues, nous nous contentons de les insérer dans une table de hachage. Pour les courtes, nous vérifions que nous ne les avons pas déjà vues, ce qui signifierait que l'entrée correspondante dans le tableau `seen[]` serait vraie (test ligne 51). Si tel est le cas, l'option est purement et simplement désactivée (ligne 52).

```
39  memset (seen, 0, sizeof (gboolean) * 256);
40  shadow_map = g_hash_table_new (g_str_hash, g_str_equal);
41
42  if (context->main_group)
43    {
44      for (i = 0; i < context->main_group->n_entries; i++)
45        {
46          entry = &context->main_group->entries[i];
47          g_hash_table_insert (shadow_map,
48                               (gpointer)entry->long_name,
49                               entry);
50
51          if (seen[(guchar)entry->short_name])
52            entry->short_name = 0;
53          else
54            seen[(guchar)entry->short_name] = TRUE;
55        }
56    }
57
```

Puis nous recommençons la même chose pour les groupes d'options. Mais est-ce vraiment le même code ? Pas tout à fait. En effet, il apparaît un test supplémentaire, si l'option longue a déjà été rencontrée. Dans ce cas, pas question de la désactiver. Mais nous la remplaçons par elle-même précédée du nom de son groupe (test lignes 65 et 66 ; remplacement ligne 67). Sinon, nous insérons l'option dans la table de hachage comme ci-dessus. Notez que le test n'a lieu que si vous n'avez pas une entrée flanquée du drapeau `G_OPTION_FLAG_NOALIAS` qui sert justement à cela.

Le principe est le même pour les options courtes, lignes 71 à 75. A la fin, la table de hachage ne nous sert plus (pas plus d'ailleurs que le tableau `seen[]`) et elle est détruite ligne 80.

```
58  list = context->groups;
59  while (list != NULL)
60    {
61      GOptionGroup *group = list->data;
62      for (i = 0; i < group->n_entries; i++)
63        {
64          entry = &group->entries[i];
65          if (g_hash_table_lookup (shadow_map,
                              entry->long_name) &&
66              !(entry->flags & G_OPTION_FLAG_
                                    NOALIAS))
67            entry->long_name = g_strdup_printf ("%s-
                %s", group->name, entry->long_name);
68          else
69            g_hash_table_insert (shadow_map,
              (gpointer)entry->long_name, entry);
70
71          if (seen[(guchar)entry->short_name] &&
72              !(entry->flags & G_OPTION_FLAG_
                                    NOALIAS))
73            entry->short_name = 0;
74          else
75            seen[(guchar)entry->short_name] = TRUE;
76        }
77      list = list->next;
78    }
79
80  g_hash_table_destroy (shadow_map);
81
```

Calcul de taille

Le code suivant calcule la taille nécessaire pour afficher la colonne de gauche, celle contenant les noms des options. Le résultat arrive dans `max_length` et indique la taille maximale d'une option. A la fin, ligne 114, 4 octets sont ajoutés pour constituer un peu d'espace entre l'option et son descriptif.

```
82  list = context->groups;
83
84  max_length = g_utf8_strlen ("-?, --help", -1);
85
86  if (list)
87    {
88      len = g_utf8_strlen ("--help-all", -1);
89      max_length = MAX (max_length, len);
90    }
91
92  if (context->main_group)
93    {
94      len = calculate_max_length (context->main_group);
95      max_length = MAX (max_length, len);
96    }
97
98  while (list != NULL)
99    {
100     GOptionGroup *group = list->data;
101
102     /* First, we check the --help-<groupname> options */
103     len = g_utf8_strlen ("--help-", -1) +
              g_utf8_strlen (group->name, -1);
104     max_length = MAX (max_length, len);
105
106     /* Then we go through the entries */
107     len = calculate_max_length (group);
108     max_length = MAX (max_length, len);
109
110     list = list->next;
111   }
112
113 /* Add a bit of padding */
114 max_length += 4;
115
```

L'affichage des options

Remarquez les trois morceaux de code. A moins qu'un groupe n'ait été indiqué sur la ligne de commande avec `--help-<group>`, le code lignes 116 à 140 est exécuté pour afficher de l'aide sur l'aide, en particulier la liste des groupes et les fameuses options `--help-<group>`.

```
116 if (!group)
117   {
118     list = context->groups;
119
120     g_print ("%s\n  -%c, --%-*s %s\n",
121              _("Help Options:"), '?',
                  max_length - 4, "help",
122              _("Show help options"));
123
124     /* We only want --help-all when there are groups */
125     if (list)
126       g_print ("  --%-*s %s\n", max_length, "help-all",
127                _("Show all help options"));
128
129     while (list)
130       {
131         GOptionGroup *group = list->data;
132
133         g_print ("  --help-%-*s %s\n", max_length - 5, group->name,
134                  TRANSLATE (group, group->help_description));
135
136         list = list->next;
137       }
138
139     g_print ("\n");
140   }
141
```

Puis, c'est le tour de deux cas, si un groupe a été demandé explicitement (lignes 142 à 150), et sinon, si nous n'avons pas de groupe principal, tous les groupes (lignes 151 à 170).

```
142 if (group)
143   {
144     /* Print a certain group */
145
146     g_print ("%s\n", TRANSLATE (group, group->description));
147     for (i = 0; i < group->n_entries; i++)
148       print_entry (group, max_length, &group->entries[i]);
149     g_print ("\n");
150   }
151 else if (!main_help)
152   {
153     /* Print all groups */
154
155     list = context->groups;
156
157     while (list)
158       {
159         GOptionGroup *group = list->data;
160
161         g_print ("%s\n", group->description);
162
163         for (i = 0; i < group->n_entries; i++)
164           if (!(group->entries[i].flags & G_OPTION_FLAG_IN_MAIN))
165             print_entry (group, max_length, &group->entries[i]);
166
167         g_print ("\n");
168         list = list->next;
169       }
170   }
171
```

Enfin, si nous avons un groupe principal, ou qu'un groupe n'a pas été indiqué spécifiquement, nous affichons le groupe principal (lignes 179 à 182) suivi des autres options (lignes 184 à 194).

```
172 /* Print application options if --help or --help-all has been
                                                    specified */
173 if (main_help || !group)
174   {
175     list = context->groups;
176
177     g_print ("%s\n", _("Application Options:"));
178
179     if (context->main_group)
180       for (i = 0; i < context->main_group->n_entries; i++)
181         print_entry (context->main_group, max_length,
182                      &context->main_group->entries[i]);
183
```

```
184       while (list != NULL)
185         {
186           GOptionGroup *group = list->data;
187
188           /* Print main entries from other groups */
189           for (i = 0; i < group->n_entries; i++)
190             if (group->entries[i].flags & G_OPTION_FLAG_IN_MAIN)
191               print_entry (group, max_length, &group->entries[i]);
192
193           list = list->next;
194         }
195
196       g_print ("\n");
197     }
198
```

A la fin, avant de quitter, si une description existe dans le contexte, nous l'affichons.

```
199   if (context->description)
200     g_print ("%s\n", TRANSLATE (context, context->description));
201
202   exit (0);
203 }
```

g_option_context_set_description()

Une telle description peut être indiquée avec `g_option_context_set_description()` :

```
01 void
02 g_option_context_set_description (GOptionContext *context,
03                                   const gchar    *description)
04 {
05   g_return_if_fail (context != NULL);
06
07   g_free (context->description);
08   context->description = g_strdup (description);
09 }
```

Cette fonction sert principalement à indiquer un message à afficher en fin de l'aide, et il est souhaitable d'y indiquer un site web où trouver plus d'informations sur le sujet, ainsi qu'une adresse électronique pour soumettre les bugs éventuels.

Conclusion

Cet article très long vous aura permis de voir comment une gestion complète des options peut se faire. Les programmeurs utilisent généralement une simple série de `printf()` dans leur code alors qu'avec de bonnes fonctions, vous pouvez arriver à un résultat plus abouti. En l'occurrence, le système que propose `GLib` permet d'avoir un affichage de l'aide toujours à jour, car il fonctionne sur la base d'un référencement. Si vous n'avez pas indiqué l'option au système, il ne peut en tenir compte pour l'analyse de la ligne de commande. Au contraire, si l'option est connue, non seulement elle pourra être analysée, mais, en plus, elle fera partie de l'aide. Enfin, GLib propose une notion de groupes d'options qui n'est pas négligeable, en particulier pour tous ceux qui programment des greffons. Ceux-ci ajoutent des options facultatives, dont la présence dépend de si le greffon est chargé ou non. Maintenant que vous avez pris connaissance de ce code, servez-vous-en !

Le mois prochain : les arbres binaires balancés de type `GTree`...

Yves Mettier,

ymettier@libertysurf.fr

Consultant Unilog/LogicaCMG et auteur de *C en action* paru chez O'Reilly

RÉFÉRENCE

▶ Le site de GTK+ : http://www.gtk.org/

Programmation sur un cluster : calculer pi avec MPI

Thibault Godouet

EN DEUX MOTS **Ne vous êtes-vous jamais demandé comment font les scientifiques et informaticiens pour faire tourner un unique calcul sur les centaines de nodes des clusters dont on entend régulièrement parler ? Ou peut-être vous demandez-vous comment faire pour utiliser toute la puissance de vos multiples machines personnelles pour effectuer un même calcul ? Ou encore voulez-vous peut-être impressionner vos amis ? ;) Quelles que soient vos motivations, cet article vous introduira les concepts d'algorithmes parallèles et de cluster, ou comment exécuter un programme sur plusieurs machines en parallèle. En guise d'exemple, nous expliquerons comment calculer pi sur un cluster avec MPI : des connaissances en langage C sont préférables.**

L'idée de combiner la puissance de plusieurs machines/processeurs pour effectuer un même calcul existe depuis longtemps. C'est nécessaire lorsqu'il n'existe pas de matériel individuellement assez puissant, et peut-être intéressant financièrement, car il est souvent moins cher d'utiliser beaucoup de matériel standard/bas de gamme qu'un gros super-calculateur ou le processeur dernier cri. Pour vous en convaincre, sachez que c'est la stratégie choisie par Google : des dizaines voire centaines de milliers de petites machines un peu partout dans le monde plutôt que quelques milliers de super-calculateurs.

Cependant, ce choix n'est pas anodin au niveau de la programmation, car les algorithmes doivent être adaptés au calcul parallèle. Paralléliser n'est pas non plus toujours intéressant en fonction du calcul à effectuer et de l'architecture matérielle, car les performances peuvent chuter drastiquement dans certains cas. Dans cet article, nous introduirons des notions sur le calcul parallèle et les illustrerons grâce à un exemple : le calcul de pi. Ce sera aussi l'occasion de découvrir MPI (*Message-Passing Interface*), qui est une bibliothèque destinée à l'envoi et réception de messages (ou données) pour les calculs distribués et les *clusters*.

Le calcul parallèle

Calcul séquentiel, calcul parallèle

Une architecture séquentielle peut être définie comme la combinaison d'un processeur avec de la mémoire (mémoire vive, disque dur...). Il s'agit d'un ordinateur personnel « standard » mono-processeur, où toutes les instructions d'un programme sont exécutées les unes après les autres.

Une architecture parallèle correspond à un ensemble de processeurs qui coopèrent pour effectuer un même calcul. Les instructions d'un programme tournant sur une telle architecture sont alors exécutées en parallèle, simultanément sur tous les processeurs.

Les architectures matérielles

Il existe un certain nombre de critères visant à comparer les architectures matérielles du point de vue du parallélisme, avec en particulier l'organisation de la mémoire et du réseau d'interconnexion.

La mémoire d'une architecture parallèle peut être distribuée, auquel cas chaque processeur dispose de sa propre mémoire avec son propre espace d'adressage. La communication entre processeurs se fait alors par « messages » envoyés par le réseau d'interconnexion. L'alternative est d'avoir un seul espace d'adressage mémoire partagé, ce qui est par exemple le cas pour les machines multiprocesseurs. Cet espace d'adressage partagé peut être implémenté à l'aide d'une banque mémoire unique, auquel cas on parle d'*Uniform Memory Access* (UMA), car tous les processeurs ont le même temps d'accès pour toute la mémoire, ou bien à l'aide de mémoire distribuée à chaque processeur, ce qui correspond à la *Non-Uniform Memory Access* (NUMA), où un processeur met moins de temps à accéder à la mémoire qui lui est rattachée qu'au reste de la mémoire (on peut aussi avoir des hiérarchies plus complexes pour les temps d'accès à la mémoire).

Pour les réseaux d'interconnexion, on distingue les réseaux statiques (ou directs), où les communications entre processeurs/nœuds se font point à point avec un câble reliant directement les deux machines, des réseaux dynamiques (ou indirects), où les connexions entre processeurs sont construites dynamiquement, par exemple par un *switch*. Dans les cas des réseaux statiques, chaque nœud peut être relié à tous les autres nœuds (nœuds « complètement connectés »), ce qui est l'idéal pour les performances, mais difficilement réalisable en pratique, ou les nœuds peuvent être reliés par des réseaux en anneau ou en grille (ou autre), auquel cas un message peut avoir à passer par plusieurs nœuds avant d'arriver à son destinataire.

Dans cet article, nous nous placerons dans le cas d'un cluster, c'est-à-dire d'une grappe d'ordinateurs « standards » reliés par un réseau commuté et fonctionnant ensemble pour réaliser un calcul. La mémoire est donc distribuée, chaque ordinateur ayant sa propre mémoire avec son propre espace d'adressage.

Performance

Paralléliser un algorithme induit une surcharge (*overhead*) dont la principale cause provient des latences dans les communications entre processeurs. En effet, il est beaucoup plus long pour un nœud d'envoyer/recevoir un message par le réseau que de l'écrire/lire dans sa mémoire. Or ce temps n'est pas directement utilisé pour faire le calcul et n'est rendu nécessaire que par le fait que l'algorithme a été parallélisé. Le temps nécessaire à l'envoi d'un message dépend de sa taille, du débit du réseau, du nombre de sauts pour arriver à sa destination, et de la méthode utilisée pour envoyer un message. Par exemple, dans le cas de la méthode *store-and-forward*, où un nœud attend d'avoir totalement reçu un message avant de le transmettre au nœud suivant, le temps d'envoi d'un message peut être calculé par la formule suivante :

$$T_{comm} = T_s + (mT_w + T_h)\,l$$

avec `Tcomm` le temps de communication, `Ts` le temps de démarrage (préparation du message et initialisation de la route, ce qui arrive une seule fois par message), `m` la taille du message, `Tw` le temps d'envoi par mot sur un lien (ce qui est directement lié au débit du lien), `Th` le temps de latence par saut (ce qui se produit pour chaque lien direct processeur à processeur), et `l` le nombre de sauts pour arriver à destination. A partir d'une telle formule, on peut calculer le temps de différentes opérations fréquemment utilisées pour faire des calculs parallèles, comme l'envoi d'un message d'un processeur à un autre, la diffusion un-vers-tous (*one-to-all*) ou tous-vers-tous (*all-to-all broadcast*).

On définit plusieurs mesures pour quantifier la performance d'un algorithme parallèle. Le Speed-Up `Sp` (avec `p` le nombre de processeurs), tout d'abord, est le facteur d'accélération du calcul en le parallélisant, défini par :

$$S_p = \frac{T_1}{T_p}$$

avec `T1` temps d'exécution du meilleur algorithme séquentiel sur un unique processeur et `Tp` le temps d'exécution de l'algorithme parallèle sur `p` processeur. L'efficacité `Ep`, quant à elle, est la fraction de temps durant laquelle un processeur fait du travail utile (par opposition à la surcharge de travail provoquée par la parallélisation de l'algorithme) et se calcule ainsi :

$$E_p = \frac{S_p}{p} = \frac{T_1}{pT_p}$$

Idéalement, chaque processeur ne fait que du travail utile et l'efficacité vaut donc 1.

Présentation de MPI

Introduction

MPI (*Message-Passing Interface*) est une spécification pour des bibliothèques destinées à l'envoi de messages : elles facilitent en particulier la communication entre processus dans un cluster et ainsi le développement de programmes destinés à tourner sur un cluster. Une telle bibliothèque contient, par exemple, des fonctions pour envoyer des données d'un processus à un autre (« Le résultat de ma partie des calculs est 33 »), qu'ils soient exécutés sur la même machine ou non, ou encore la synchronisation entre les processus (« Je suis prêt, tu peux m'envoyer les données »).

Il existe plusieurs implémentations de MPI, avec en particulier LAM et MPICH. Nous utiliserons ce dernier dans cet article. En plus de la bibliothèque MPI elle-même, MPICH inclut des logiciels destinés à la compilation ou l'exécution des programmes parallélisés. C'est grâce à eux que vous pourrez lancer un programme sur un ensemble de machines presque comme si vous le lanciez sur une seule.

MPI est assez similaire à PVM (*Parallel Virtual Machine*) dans ses objectifs et sa réalisation. Cependant, MPI a certaines particularités intéressantes, comme une spécification complète et plusieurs implémentations indépendantes, le déterminisme ou une communication entre processus plus avancée (voir **[2]**). MOSIX (voir **[6]**), quant à lui, a une approche un peu différente. En effet, MOSIX a pour but de faire apparaître, à un programme, un cluster comme une seule machine multiprocesseur. Il n'est notamment pas nécessaire d'utiliser une API spécifique pour programmer le logiciel pour le cluster – les classiques *threads* suffisent –, ni même de compiler ce logiciel avec des outils MOSIX. Par contre, MOSIX requiert un noyau particulier pour arriver à cette transparence du point de vue programme utilisateur. MOSIX gère aussi les migrations de threads/processus entre nœuds du cluster de façon à équilibrer la charge automatiquement. MOSIX requiert donc une installation et configuration plus longue et complexe, mais ne nécessite pas de modifications des applications – à condition qu'elles soient déjà multithreadées. MOSIX paraît plus adapté lorsque les machines du cluster peuvent aussi être utilisées indépendamment du cluster, par exemple pour faire tourner des calculs coûteux sur les ordinateurs personnels d'une entreprise lorsqu'ils sont inutilisés, tandis que PVM et MPI sont plutôt destinés aux clusters composés de machines dédiées uniquement, et qui font les calculs les uns après les autres (mode « batch »). Enfin, MOSIX est nettement moins portable que MPI, et n'est implémenté que sur les systèmes Linux.

Pi est un nombre irrationnel, c'est-à-dire qu'il n'est pas le rapport de deux nombres entiers naturels. Il est également appelé « constante d'Archimède ».

Installer MPICH

MPICH est disponible sous de nombreuses plateformes : Linux (paquets pour Debian/Ubuntu, Fedora, Slackware...), ou encore FreeBSD et même Windows, tant en 32 qu'en 64 bits.

Le plus simple pour installer MPICH sera certainement d'utiliser votre gestionnaire

de paquet préféré. A titre d'exemple, les paquets à installer sous une Debian sont : `libmpich1.0`, `libmpich1.0-dev`, `mpich-bin` et (optionnel) `mpi-doc`. Si vous le souhaitez, vous pouvez aussi le compiler à partir des sources (voir **[4]**). Vous n'avez pas besoin d'installer MPICH sur toutes vos machines, mais seulement sur la machine « serveur », autrement dit sur celle à partir de laquelle vous lancerez l'exécution d'un programme avec `mpirun` (voir plus loin).

Si vous êtes curieux, vous avez peut-être remarqué que nous utilisons MPICH et non pas MPICH2, qui est une nouvelle implémentation de MPI-1, version d'origine de MPI sur laquelle nous nous concentrons, et son extension MPI-2. En effet, au moment où est écrit cet article, MPICH semble plus disponible que MPICH2 (il n'y a pas de paquet MPICH2 inclus dans Debian par exemple), et nous n'avons pas besoin des extensions de la norme MPI-2 qui n'est pas implémentée dans MPICH. Cet article devrait de toute façon être facilement adaptable à MPICH2 qui supporte toujours MPI-1.

Selon votre distribution ou votre choix, MPICH utilisera rsh ou ssh pour lancer les processus sur les machines distantes. Il conviendra donc d'installer et configurer correctement rsh ou ssh sur toutes les machines. Si vous utilisez ssh, vous pouvez vous reporter à **[5]** pour configurer vos machines de façon à ne pas avoir à taper les mots de passe à chaque exécution d'un programme par MPICH.

Pi est un nombre transcendant. Il n'existe pas de polynôme à coefficients entiers ou rationnels dont pi soit une racine. Cette caractéristique établit l'impossibilité de résoudre le problème de la quadrature du cercle : il est impossible de construire, à l'aide de la règle et du compas, un carré dont la surface soit rigoureusement égale à la surface d'un cercle donné.

Calculer pi

Algorithme pour calculer pi

Il existe de nombreuses méthodes pour calculer pi, avec certaines plus originales et distrayantes que d'autres. Dans le cas présent, nous allons utiliser une méthode assez simple et facilement parallélisable. Ne vous inquiétez pas si les détails des calculs qui suivent vous échappent : ils n'ont aucune importance pour l'objet de cet article et ne sont là que pour satisfaire la curiosité des lecteurs amateurs de mathématiques. Seule la dernière formule de cette partie sera réutilisée dans l'algorithme.

On remarque que l'intégrale suivante vaut pi :

$$\begin{aligned} f(x) &= \frac{4}{1+x^2} \\ \int_0^1 f(x)dx &= 4\int_0^1 \frac{1}{1+x^2}dx \\ &= 4\left(\left[\arctan(x)\right]_0^1\right) \\ &= 4\left(\arctan(1)-\arctan(0)\right) \\ &= 4\left(\frac{\pi}{4}-0\right) \\ &= \pi \end{aligned}$$

Nous allons exprimer cette intégrale grâce à un découpage en trapèzes. En effet, on a dans le cas général :

$$\int_a^b f(x)dx = \lim_{n\to\infty}\sum_{i=1}^{n}\left(f(x_i)+f(x_{i+1})\right)\frac{h}{2}$$

avec :

$$h = \frac{b-a}{n}$$

et :

$$x_i = (i-1)h + a$$

L'approximation (nécessaire vu qu'on ne peut pas calculer numériquement une somme infinie sur une machine !) consistera à utiliser un `n` fini, et non plus le faire tendre vers l'infini. Plus `n` sera grand, plus l'approximation sera bonne :

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n}\left(f(x_i)+f(x_{i+1})\right)\frac{h}{2}$$

En appliquant cette formule à notre cas particulier, on obtient :

$$\int_0^1 f(x)dx \approx \frac{1}{2}\sum_{i=1}^{n}\left(f\left(\frac{i-1}{n}\right)+f\left(\frac{i}{n}\right)\right)$$

Notre programme va donc calculer cette somme finie pour un *n* grand, ce qui nous donnera comme résultat une (bonne) approximation de pi.

Implémentation sur une machine

De façon à avoir une bonne précision, nous utiliserons des `doubles` et non des `floats`. On affichera aussi 12 décimales pour pi, ce qu'on obtient grâce à `%.12f` dans les `printf()`. Le code n'a pas de difficulté particulière : on lit le nombre de trapèzes (i. e. le `n` dans l'algorithme ci-dessus) à utiliser du premier argument de la ligne de commande, on utilise une boucle `for` et la fonction `f()` pour calculer la somme vue dans la partie précédente, puis la valeur de pi est affichée à la fin du programme :

```
/*
 * pi - calculer pi
 * Copyright 2006 Thibault GODOUET
 */
/*
 * A compiler avec:
 * gcc -Wall -O2 pi-seq.c -o pi-seq
 * puis lancer avec par exemple:
 * ./pi-seq 100000000
 */
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
/* Fonction à intégrer
 * pour calculer pi: */
/* NOTE: le "inline" n'est là que pour
```

```
 * (peut-être!) améliorer
 * les performances et peut être enlevé
 * sans problème */
inline
double
f(double x)
{
  return ( 4.0 / (1.0 + x*x) );
}
int
main(int argc, char **argv)
{
  /* nombre de trapèzes: */
  long long int num_trap = 0;
  long long int i = 0;
  double x = 0;
  double h = 0;
  double sum = 0;
  /* En cas de problème (deadlock,
   * boucle infinie, etc), arrêter le
   * programme au bout de 60 secondes: */
  alarm(600);
  /* Lire le nombre de trapèzes à
   * utiliser pour le calcul de
   * l'intégrale */
  num_trap = atoll(argv[1]);
  printf("Nombre de trapèzes: %Ld\n",
          num_trap);
  /* Calculer la somme approximant
   * l'intégrale et donc pi: */
  h = 1 / (long double)num_trap;
  x = 0;
  sum = 0;
  for ( i = 2; i < num_trap; i++) {
    sum += f(x) * h;
    x += h;
  }
  sum += (f(0) + (f(1))) * h
    / ( (long double) 2);
  printf("pi=%.12f\n", sum);
  return 0;
}
```

Reste à compiler et à exécuter, par exemple avec 100 000 000 de trapèzes :

```
$ gcc -Wall -O2 pi-seq.c -o pi-seq
$ time ./pi-seq 100000000
Nombre de trapèzes: 100000000
Pi=3.141592653590
real    0m1,490s
user    0m1.472s
sys     0m0.000s
```

Comme vous pouvez le constater, le temps d'exécution est d'environ une seconde et demi sur l'ordinateur de test : ce n'est pas forcément très long, mais on le sent déjà. Vous pouvez vous amuser à rajouter quelques zéros pour voir le temps que ça prendra !

Quant à la précision, elle est déjà très bonne, vu que les 12 premières décimales de pi que nous avons calculées sont bonnes (en fait, elles finissent normalement par 89, mais vu que la décimale suivante est un 7, le tout est arrondi à 90).

AA REMARQUE

Il se peut selon les machines que vous utilisez que la précision ne soit pas si bonne. En effet, certaines erreurs d'arrondi peuvent se produire lorsqu'on additionne un très grand nombre de *floats*/doubles entre eux, en particulier quand on additionne un nombre float/double avec un autre qui est beaucoup plus petit que lui : c'est ce qu'on fait ici lors du `sum += f(x)*h`, car `sum` est beaucoup plus grand que `f(x)*h` après un certain nombre d'itérations. Nous avons préféré garder notre approche simpliste pour ne pas noyer le lecteur qui découvre déjà le calcul parallèle et MPI, mais pour éviter toute erreur d'arrondi, il faudrait additionner les `f(x)` deux à deux, puis additionner les résultats deux à deux, et ainsi de suite jusqu'à trouver la somme finale, ce qui crée un arbre de profondeur `log2(num_trap)`. Le lecteur intéressé trouvera plus d'information sur ce sujet en **[7]**.

Paralléliser le calcul de pi avec MPI

Introduction

Nous allons paralléliser cet algorithme en découpant la grosse somme en plusieurs sous-sommes de même taille : chaque sous-somme sera exécutée sur un nœud distinct. Ensuite, un nœud maître récupérera les valeurs des sous-sommes calculées par les différents nœuds, et les additionnera pour obtenir la valeur de pi. Il est important que chaque nœud ait le même temps de calcul et donc la même quantité de travail si les nœuds sont identiques, car le nœud maître devra attendre tous les résultats intermédiaires avant de pouvoir fournir le résultat final. Aussi, si une machine finit son travail 10 secondes avant les autres, alors ces 10 secondes auraient pu (dû !) être utilisées pour finir le calcul général plus rapidement. Plus généralement, la règle est d'affecter autant de processus à un nœud qu'il a de processeurs, à condition bien entendu que tous les processus aient la même « charge de travail » et que tous les processeurs soient identiques. Alors, pourquoi la somme finale est-elle faite par une seule machine me demandez-vous ? Tout simplement parce que, dans ce cas précis, la somme finale est très rapide à faire, comparée aux calculs des sous-sommes. Aussi il ne paraît pas nécessaire de rendre le code sensiblement plus compliqué pour un gain très minime.

Journée de pi : la date du 14 mars, écrit 3/14 au format américain à 1h59 (parce que 3,14159) est généralement utilisée pour célébrer la constante. On notera que le 14 mars est également le jour de l'anniversaire d'Albert Einstein.

Programmer avec MPI

MPI est une bibliothèque contenant plus de 120 fonctions, mais seules quelques-unes (moins d'une dizaine) sont nécessaires pour un programme simple comme celui que nous écrivons. Nous nous contenterons ici d'une courte introduction à la programmation avec MPI : le lecteur souhaitant approfondir ce sujet pourra se reporter à **[3]**.

Pour utiliser MPI, un programme doit inclure le *header* de MPI :

```
#include "mpi.h"
```

Ensuite, `MPI_Init(argc, argv)` doit être appelé avant toute autre fonction de la bibliothèque MPI, et `MPI_Finalize(void)` doit être appelé à la fin de l'exécution du logiciel pour libérer les ressources.

Les communications dans un cluster MPI sont organisées par groupes de communication. Cette fonctionnalité peut s'avérer très utile pour des programmes un peu complexes, mais, dans notre cas, nous nous contenterons du groupe de communication `MPI_COMM_WORLD` qui, vous l'aurez deviné, contient tous les nœuds du cluster.

On peut obtenir la taille du groupe de communication (et donc du cluster avec `MPI_COMM_WORLD`) en appelant la fonction `int MPI_Comm_size(MPI_Comm, int *size)`, par exemple :

```
MPI_Comm_size(MPI_COMM_WORLD, &size);
```

Chaque nœud du cluster MPI a un rang dans son groupe de communication, et il s'obtient en appelant la fonction `int MPI_Comm_rank(MPI_Comm comm, int *rank)` comme dans :

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

Pour envoyer un message de façon bloquante, `int MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)` est votre ami : `buf` contient les données à envoyer, `rank` est le rang du nœud (comme obtenu par `MPI_Comm_rank()`) à qui on envoie la donnée et `tag` permet de savoir à quoi correspond cette donnée. Par exemple :

```
#define TAG_NB_TRAPEZES 1000
int nb_trapezes = 100000;
MPI_Send(&nb_trapezes, 1, MPI_INT, 0,
        TAG_NB_TRAPEZES, MPI_COMM_WORD);
```

La fonction réciproque, pour recevoir un message de façon bloquante, est `int MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status);` comme dans :

```
int nb_trapezes;
MPI_Status status;
MPI_Recv(&nb_trapezes, 1, MPI_INT,
     MPI_ANY_SOURCE, TAG_NB_TRAPEZES,
     MPI_COMM_WORLD, &status);
```

`status` contenant des informations sur le message reçu :

```
status.count      = message length
status.MPI_SOURCE = message sender
status.MPI_TAG    = message tag
```

A noter qu'on peut aussi utiliser `MPI_ANY_TAG` pour accepter des messages quel que soit leur tag.

Il est de plus possible d'envoyer une donnée à tous les nœuds, en utilisant `int MPI_Bcast(void* buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)` : selon que le nœud en question envoie (son rang est alors égal à la valeur de `root`) ou reçoit, `buffer` contient la valeur à envoyer ou est l'endroit où sera écrite la valeur reçue (pensez à allouer la mémoire nécessaire !). Par exemple :

```
long long int num_trap = 100000;
 MPI_Bcast((void *) &num_trap, 1,
    MPI_LONG_LONG_INT, 0,
    MPI_COMM_WORLD);
```

Pour récupérer sur un nœud une valeur de chacun des nœuds du cluster, il convient d'utiliser `int MPI_Gather(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)`. Les noms des arguments sont assez explicites. A noter que `recvcount` correspond au nombre de données reçues par nœud du cluster, pas du nombre total pour tout le cluster. Voici un exemple d'utilisation :

```
int size;
int rank;
double *sum_buf = NULL;
double sum = 0;
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
if ( rank == 0 )
  sum_buf = malloc(sizeof(double)
        * size);
MPI_Gather((void *)&sum, 1,
          MPI_DOUBLE, sum_buf, 1,
          MPI_DOUBLE, 0,
          MPI_COMM_WORLD);
```

Avec ces connaissances, nous pouvons maintenant passer à l'implémentation de notre programme calculant pi.

Implémentation

Nous reprenons le code du programme séquentiel et le modifions pour obtenir le code suivant :

```
/*
 * pi - calculer pi
 *      sur un cluster avec MPI
 * Copyright 2006 Thibault GODOUET
 */

/*
 * A compiler avec:
 * mpicc -Wall -O2 pi-parallel.c \
   -o pi-parallel
 * puis lancer avec par exemple:
 * mpirun -machinefile machines \
   -np 1 pi-parallel 100000000
 */

#include <stdarg.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

/* Inclure les déclarations
 * des fonctions MPI */
#include "mpi.h"

/* Nombre de noeuds dans le cluster,
 * et rang de ce noeud */
int size, rank;
/* Pour calculer le temps de calcul: */
double start_time, end_time;

/* printf() avec le rang du noeud
 * affiché au début de la ligne */
void
xprintf(char *fmt, ...)
{

  va_list args;

  va_start(args, fmt);
  printf("noeud %d: ", rank);
  vprintf(fmt, args);
}

/* Sortir proprement du programme,
 * et afficher le temps de calcul */
void
xexit(int code)
{
  if ( rank == 0 ) {
    end_time = MPI_Wtime();
    xprintf("TEMPS DE CALCUL: "
      "%f secondes\n",
      end_time - start_time);
  }

  MPI_Finalize();
  exit(code);
}

/* Fonction à intégrer
 * pour calculer pi: */
/* NOTE: le "inline" n'est là que pour
 * (peut-être!) améliorer
 * les performances et peut être enlever
 * sans problème */
inline
double
f(double x)
{
  return (4.0 / (1.0 + x*x));
}

/* Calcul de la partie de la somme
 * approximant l'intégrale de f()
 * à faire par ce noeud */
double
sub_sum(long long int start_i,
        long long int stop_i, double h)
{
  double f_xi, f_xi1;
  long long int i = 0;
  double sum = 0;
  double x = 0;

  xprintf("Sous-somme de %Ld à %Ld\n",
            start_i, stop_i);
  sum = 0;
  x = h*(start_i-1);
  f_xi1 = f(x);
  for ( i = start_i; i <= stop_i; i++ ) {
    f_xi = f_xi1;
    f_xi1 = f(x+h);
    sum += (f_xi+f_xi1);
    x += h;
  }
  sum *= h/2.0;
  return sum;
}

int
main(int argc, char **argv)
{

  /* nombre de trapèzes: */
  long long int num_trap = 0;
  long long int i = 0,
    start_i = 0, stop_i = 0;
  double h = 0;
  double sum = 0;
  /* buffer pour récupérer
   * les sous-sommes des différents
   * noeuds: */
  double *sum_buf = NULL;

  size = rank = -1;

  /* En cas de problème (deadlock,
   * boucle infinie, etc), arrêter le
   * programme au bout de 60 secondes: */
alarm(600);

  MPI_Init(&argc, &argv);

  MPI_Barrier(MPI_COMM_WORLD);
  start_time = MPI_Wtime();

  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  if (rank == 0) {
    /* Noeud maître*/

    xprintf("Taille du cluster: %d\n",
            size);

    /* Lire le nombre de trapèzes à
     * utiliser pour le calcul de
     * l'intégrale */
    num_trap = atoll(argv[1]);
    xprintf("Nombre de trapèzes: %Ld\n",
          num_trap);
  }

  /* Envoyer/recevoir le nombre
   * de trapèzes */
  MPI_Bcast((void *) &num_trap, 1,
            MPI_LONG_LONG_INT, 0,
            MPI_COMM_WORLD);

  /* Calculer notre partie
   * de la somme: */

  h = ( 1.0 - 0.0 ) / (double)num_trap;
  start_i = 1 + num_trap * rank / size;
  stop_i = num_trap * (rank + 1) / size;

  sum = sub_sum(start_i, stop_i, h);

  /* récupérer toutes les valeurs
   * des sous-sommes */

  if ( rank == 0 )
    /* Noeud maître: c'est nous qui
     * récupérons les valeurs, nous
```

```
    * devons donc allouer de la mémoire
    * à cette fin */
   sum_buf = malloc(sizeof(double)
                    * size);

  MPI_Gather((void *)&sum, 1,
             MPI_DOUBLE, sum_buf, 1,
             MPI_DOUBLE, 0,
             MPI_COMM_WORLD);

  if ( rank == 0 ) {

    /* noeud maître: sommer
     * les sous-sommes pour obtenir
     * la valeur de pi */

    sum = 0;
    for (i=0 ; i < size ; i++ ) {
      xprintf("somme=%.12f pour le "
              "noeud %d\n",
              sum_buf[i], i);
      sum += sum_buf[i];
    }

    xprintf("pi=%.12f\n", sum);

  }
  xexit(0);

  /* Nous n'arrivons jamais ici,
   * mais la ligne suivante est
   * nécessaire pour éviter un warning
   */
  return 0;

}
```

Pour clarifier le code et la sortie des exécutions, nous avons créé deux nouvelles fonctions, `xprintf()` et `xexit()`. `xprintf()` se comporte comme `printf()`, mais en rajoutant le numéro du nœud qui a imprimé un message, tandis que `xexit()` quitte proprement le programme en appelant `MPI_Finalize()`.

A noter aussi que le temps d'exécution est chronométré grâce à deux appels à `MPI_Wtime()`, destiné à cet effet. Le temps chronométré est le véritable temps d'exécution, en excluant le temps d'initialisation de MPI et de lancement des processus sur les machines distantes : ce temps ne nous intéresse pas pour évaluer la qualité de notre algorithme et est généralement négligeable pour un gros calcul, mais ce n'est pas forcément le cas pour nos tests. Immédiatement avant le début du chronométrage, on appelle `MPI_Barrier()`, qui sert à synchroniser les processus : la fonction ne retourne que quand elle a été appelée par tous les processus du cluster.

On a aussi introduit une fonction `sub_sum()` qui s'occupe de calculer la sous-somme d'un nœud donné. Le calcul a été un peu optimisé, par exemple en ne multipliant par h/2 qu'une seule fois et pas à chaque itération de la boucle. Cela peut paraître assez superficiel, mais ne faire qu'une seule multiplication au lieu de plusieurs millions finit par se ressentir ! De même, on a choisi de calculer la somme de la façon la plus proche possible de la formule mathématique vue plus haut, principalement pour clarifier les choses et ainsi éviter les erreurs d'indices, mais aussi pour limiter les erreurs d'arrondi évoqués plus haut (de façon imparfaite pour simplifier, voir la note plus haut à ce sujet pour plus de détails).

En début de programme, le nœud maître (de rang 0) lit le nombre de trapèzes à utiliser de la ligne de commande, et le communique à tous les nœuds grâce à `MPI_Bcast()`. S'en suivent les calculs sur chaque nœud, puis le nœud maître récupère les valeurs de tous les nœuds avant de les additionner et d'afficher le résultat de pi.

Compilation et exécution

Comme évoqué précédemment, un ensemble de programmes pour effectuer la compilation et l'exécution est fourni avec MPI. Pour compiler, nous utiliserons `mpicc` avec les mêmes arguments que `gcc` : `mpicc` s'occupera de lier le binaire avec la bibliothèque de façon transparente. On lance ensuite le calcul à l'aide de `mpirun`, en indiquant un fichier `machines` contenant les noms des machines à utiliser pour le calcul (un par ligne, et, si besoin, le nom complet avec le nom de domaine), et le nombre de processus parallèles qu'on veut lancer (argument `-np`), suivi du nom du programme à exécuter et des arguments de ce programme : `mpirun` lancera lui-même les processus sur les machines distantes. On pensera bien à vérifier que les machines sont non utilisées avant de lancer nos calculs, histoire que les temps de calcul ne soient pas faussés.

```
$ mpicc -Wall -O2 pi-parallel.c -o pi-parallel

$ mpirun -machinefile machines  -np 1 ./pi-parallel 10000000
noeud 0: Taille du cluster: 1
noeud 0: Nombre de trapèzes: 10000000
noeud 0: Sous-somme de 1 à 10000000
noeud 0: somme=3.141592653590 pour le noeud 0
noeud 0: Pi=3.141592653590
noeud 0: TEMPS DE CALCUL: 1.441362 secondes

$ mpirun -machinefile machines  -np 2 ./pi-parallel 10000000
thib@machine2's password:
noeud 0: Taille du cluster: 2
noeud 0: Nombre de trapèzes: 10000000
noeud 0: Sous-somme de 1 à 5000000
noeud 0: somme=1.854590436003 pour le noeud 0
noeud 0: somme=1.287002217587 pour le noeud 1
noeud 0: Pi=3.141592653590
noeud 0: TEMPS DE CALCUL: 0.723543 secondes
noeud 1: Sous-somme de 5000001 à 10000000

$ mpirun -machinefile machines  -np 4 ./pi-parallel 10000000
thib@machine2's password:
thib@machine3's password:
thib@machine4's password:
noeud 0: Taille du cluster: 4
noeud 0: Nombre de trapèzes: 10000000
noeud 0: Sous-somme de 1 à 2500000
noeud 0: somme=0.979914652507 pour le noeud 0
noeud 0: somme=0.874675783496 pour le noeud 1
noeud 0: somme=0.719413999170 pour le noeud 2
noeud 0: somme=0.567588218417 pour le noeud 3
```

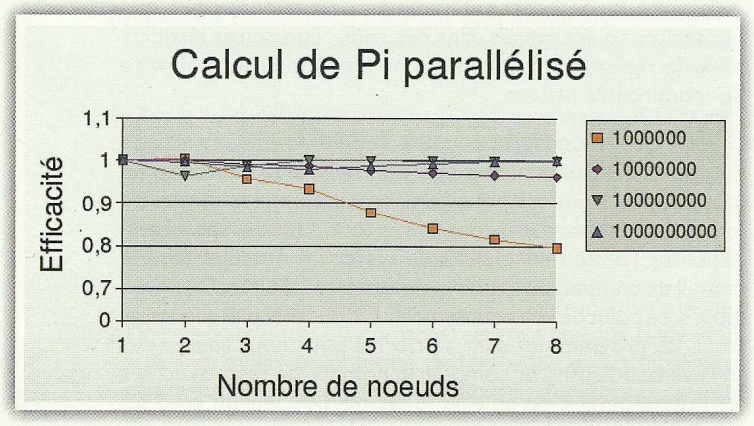
```
noeud 0: Pi=3.141592653590
noeud 0: TEMPS DE CALCUL: 0.365046 secondes
noeud 2: Sous-somme de 5000001 à 7500000
noeud 3: Sous-somme de 7500001 à 10000000
noeud 1: Sous-somme de 2500001 à 5000000
```

Comme vous pouvez le constater, les sorties des différents nœuds arrivent de façon désordonnée. Plus précisément, les sorties d'un unique nœud sont bien dans l'ordre chronologique, mais une sortie du nœud 2 peut être affichée après une sortie du nœud 0 même si elle a été émise avant. Si cela posait un problème, on pourrait utiliser des `fflush(stdout)` par exemple dans `xprintf()` pour forcer l'envoi et l'affichage des sorties au fur et à mesure. Les performances pourraient cependant en être dégradées.

Performance et analyse

L'algorithme de calcul de pi étudié est hautement parallélisable : en effet, comme vous avez pu le constater grâce aux exécutions ci-dessus, le temps de calcul est divisé par deux quand on utilise deux fois plus de processeurs. C'est ce à quoi on s'attend, mais c'est loin d'être évident ! Le principal problème qui empêche cela est la communication entre processeurs. Ici, elle est très limitée : on envoie le nombre de trapèzes au début, et on envoie le résultat de chaque sous-somme à la fin. Au final, on a de l'ordre de 2 messages par nœud (selon l'implémentation du `MPI_Gather()`) de chacun la taille d'un `double` ou d'un `long long int`. Pour d'autres applications, comme une multiplication de matrice par un vecteur, la quantité de communication entre nœuds, tant en nombre qu'en taille, est beaucoup plus importante. Or, cette communication se fait par le biais d'un réseau qui est très lent comparé à un simple accès disque et encore plus un accès mémoire.

Ceci est illustré par le graphe représentant l'efficacité de l'algorithme en fonction du nombre de nœuds et du nombre de trapèzes utilisés :



Comme vous pouvez le constater, l'efficacité est très bonne (très proche de 1) dans tous les cas, sauf pour 1 000 000 de trapèzes où elle se dégrade rapidement quand on augmente le nombre de processeurs. En effet, la durée des calculs est alors réduite par rapport aux cas où il y a plus de trapèzes, alors que chaque nœud passe toujours autant de temps à communiquer avec les autres nœuds : le temps de calcul utile sur le temps total est plus faible, donc l'efficacité est elle-même plus faible.

Conclusion

Utiliser un cluster pour faire de longs calculs est souvent intéressant financièrement par rapport à d'autres solutions et peut être indispensable quand aucun matériel existant ne peut individuellement faire le calcul dans un temps raisonnable.

Cependant, tous les problèmes ne se parallélisent pas aussi bien les uns que les autres sur un cluster. En effet, les coûts (en termes de temps) des communications par le réseau sur un cluster sont très importants comparés à des accès disque ou mémoire, ce qui explique que les calculs nécessitant beaucoup de communications ne sont pas faits aussi efficacement sur un cluster qu'on pourrait l'imaginer au premier abord.

MPI facilite beaucoup le développement de programmes destinés à des clusters, tant par sa bibliothèque que par ses outils de compilation et d'exécution, tout cela pour le modeste prix de l'apprentissage de quelques fonctions de son API.

Thibault Godouet,

thibault.godouet@m4tp.org

BIBLIOGRAPHIE

- **[1]** *The Message Passing Interface (MPI) standard,* http://www-unix.mcs.anl.gov/mpi
- **[2]** *Top 10 reasons to prefer MPI over PVM,* http://www.lam-mpi.org/mpi/mpi_top10.php
- **[3]** *The Message Passing Interface (MPI) standard,* http://www-unix.mcs.anl.gov/mpi/
- **[4]** *MPICH Documents,* http://www-unix.mcs.anl.gov/mpi/mpich1/docs.html
- **[5]** *How to get mpich 1.2.5 working with ssh,* http://www.me.berkeley.edu/~kirpekar/sshMpich.htm
- **[6]** *About MOSIX,* http://www.mosix.org/txt_about.html
- **[7]** *What Every Computer Scientist Should Know About Floating-Point Arithmetic,* http://docs.sun.com/source/806-3568/ncg_goldberg.html

→ Tests et Perl – Bonnes pratiques

Sébastien Aperghis-Tramoni & Philippe Blayo

EN DEUX MOTS **Le précédent article vous a décrit les bases du mécanisme général de test en Perl, le protocole TAP et le module associé Test::Harness, ainsi que le module de test le plus courant et le plus utile pour débuter, Test::More. Nous allons maintenant voir comment bien gérer ses tests, et étudier l'utilisation de quelques-uns des nombreux modules disponibles sur le CPAN pour simplifier la mise en œuvre de sémantiques complexes.**

Bonne gestion des tests

Voyons d'abord quelques bonnes pratiques de gestion des scripts de tests.

Qualité du code

Premier point important à garder à l'esprit, le code des scripts de tests est du code Perl comme les autres. Il n'y a donc pas de raison pour qu'il soit plus mal écrit qu'un autre. En fait, comme c'est typiquement du code que la plupart des personnes n'aiment pas écrire ni relire, il est préférable, encore plus que pour le reste du code, qu'il soit écrit de manière propre, lisible et facile à maintenir. Qu'est-ce que cela signifie ? Que le code devrait dans la mesure du possible être écrit en respectant les règles de bonnes pratiques que vous avez l'habitude d'utiliser, par exemple que le code doit passer avec les structures activées. Autre conseil, éviter de réinventer les roues qui existent déjà. Cela se traduit par l'utilisation de modules qui fournissent ces roues, modules qui seront justement présentés dans la suite de cet article.

Compartimenter ses tests

Une question qui peut venir à l'esprit est de savoir comment répartir ses tests. Vaut-il mieux les mettre tous dans un seul script ou au contraire les répartir le plus possible ? La réponse est que cela dépend :-)

En particulier, cela dépend du code testé et du nombre de fonctions ou fonctionnalités à tester. S'il ne possède qu'un nombre restreint de fonctions ou un périmètre fonctionnel très délimité, l'ensemble des tests peuvent alors tenir dans un seul script court. Par exemple, le module `XSLoader`, qui fait partie de la distribution standard de Perl, contient l'ensemble de ses tests fonctionnels dans le script `t/XSLoader.t`. La raison en est que ce module, dont le rôle est de servir d'interface simplifiée à `DynaLoader`, le chargeur dynamique de bibliothèques externes, ne fournit qu'une seule fonction, `load()`. Comme celle-ci ne s'utilise pas de dix manières différentes, on a vite fait de tout tester !

Quand le code à tester comporte un périmètre fonctionnel plus important, la solution la plus pratique est d'écrire un script par fonction, groupe de fonctions, fonctionnalité ou type de données à traiter, suivant les spécificités du code en question. C'est généralement la méthode retenue par beaucoup de développeurs publiant leurs modules sur le CPAN. Regardons quelques exemples plus en détail.

Dans le cas du module `Parse::Syslog::Mail`, assez simple puisqu'il ne fournit que deux méthodes, les scripts se répartissent les tâches de la manière suivante :

- `01api.t` se contente de vérifier si les méthodes de ce module fonctionnent comme prévu.
- `03diag.t` vérifie les messages d'erreur et d'avertissement que le module peut produire.
- `10fields.t` et `11values.t` vérifient que le module analyse correctement des ensembles de données (fournis avec le module) ; le premier vérifie mécaniquement que les champs ont bien été isolés, le second que les valeurs renvoyées correspondent à ce qui était attendu.

Dans le cas d'un module plus complexe comme `Net::Proxy`, la suite de tests doit aussi réaliser des tests plus élaborés.

- Les premiers scripts, de `10proxy_new.t` à `14proxy_debug.t` se contentent de tester l'API du module, en invoquant le constructeur de différentes manières.
- Les scripts suivants, en particulier les `30tcp_tcp.t` à `33dual.t`, sont plus sophistiqués et créent des clients et serveurs réseau afin de tester les fonctionnalités de *proxying* proprement dites du module.

Enfin, pour les modules poids lourds comme `LWP` ou `POE`, leur suite de tests est suffisamment importante pour que les scripts soient rangés dans des sous-répertoires distincts dans le répertoire `t/`, nommés en fonction du domaine de fonctionnalités testées.

Pourquoi compter ses tests ?

Un point qui peut sembler très contraignant à beaucoup de développeurs (il l'est pour l'un des auteurs) est de devoir compter les tests d'un script. Après tout, il est bien plus facile d'utiliser l'option `no_plan` et de laisser les modules faire le travail de comptage des points de test à notre place. De prime abord, ce point de vue peut sembler raisonnable puisque, après tout, les programmes sont a priori là pour nous libérer des tâches fastidieuses, et compter le nombre de tests exécutés en fait partie. Et c'est en effet le cas, mais seulement pour les tests *exécutés* : à tout moment, on peut savoir combien de tests ont été exécutés (que ce soit avec succès ou échec). Mais le script de test n'a aucun moyen de savoir combien de tests *restent à exécuter*. Cette information ne peut lui être fournie que de l'extérieur, par la personne qui écrit le script.

`Test::More`

À noter que lorsqu'on parle des fonctionnalités de `Test::More`, celles-ci sont généralement héritées de `Test::Builder`

qui est le module d'architecture sur lequel il s'appuie, et qu'elles s'appliquent donc en règle générale aux modules qui dérivent eux aussi de `Test::Builder`.

Le programmeur paresseux va se demander quelle est l'importance de connaître à l'avance le nombre de tests à exécuter. La réponse est qu'il s'agit d'une contrainte de cohérence du script, permettant de vérifier son bon déroulement. La raison technique se trouve dans la description du protocole TAP de l'article précédent. Dans le cas normal, quand le plan est déclaré en début de session, l'interpréteur TAP (dans le cas général, il s'agit de `Test::Harness`) sait combien de tests doivent être exécutés par le script. Si celui-ci meurt en cours d'exécution, l'interpréteur TAP est donc en mesure de s'en rendre compte, car il verra une interruption du flux avant le nombre prévu :

```
1..51
ok 1
ok 2
...
ok 22
ok 23
```

Ici, le script s'arrête après le point 23 alors qu'il aurait dû en exécuter 51. De plus, `Test::Harness` va recueillir le statut de sortie du script et le décoder pour essayer d'en savoir plus. `Test::More`, par exemple, sort avec comme valeur de statut le nombre de tests qui ont échoué.

On peut trouver là une nouvelle raison de penser qu'il n'y a pas besoin de s'ennuyer avec le comptage des tests, puisque, en cas d'erreur, `Test::More` va positionner le statut avec une valeur d'erreur qui sera détectée par `Test::Harness`. Sauf qu'il y a plusieurs problèmes qui font que ce n'est pas vrai partout, et que cela va de moins en moins le devenir à l'avenir. Car si `Test::Harness` a actuellement une forte dépendance par rapport au statut de sortie du script de test, cela provient surtout de l'histoire de Perl, qui est un langage né sur Unix. Or Perl est disponible sur un grand nombre de plateformes, et certains systèmes n'ont pas la même notion de statut de sortie, voire pas de statut du tout. Ce mécanisme fonctionne bien sous Unix, mais il n'est pas garanti ailleurs.

Portabilité

Après tout, cela ne devrait surprendre personne, Perl étant un langage contenant un nombre non négligeable de fonctions n'ayant vraiment de sens que sous Unix, et dont certaines sont très difficiles, voire impossibles à émuler sur des systèmes ne possédant pas de fonctions équivalentes, comme par exemple `fork()`, `chroot()`, `chmod()`, `chown()`, `kill()`, `link()`, les fonctions IPC System V, et d'autres encore.

De plus, deux autres phénomènes tendent à pousser à l'abandon, à plus ou moins court terme, de l'utilisation du statut comme déterminant primaire du succès ou de l'échec d'un script. Le premier est l'utilisation de `Test::Harness` pour tester des programmes écrits dans d'autres langages de programmation. Suivant les langages considérés, le positionnement du statut de sortie peut ou non être supporté, et, même quand il l'est, ne pas être positionné pour des raisons culturelles (on peut notamment penser à Java).

Le second est que le flux TAP peut ne pas être généré par un script directement contrôlé par `Test::Harness`, mais provenir d'une source extérieure, par exemple au travers du réseau en utilisant HTTP comme mécanisme de transfert. Il n'y aura alors pas de notion de statut du tout.

Enfin et plus fondamentalement, le statut de retour du script ne fait pas partie de la spécification du protocole TAP. Il s'agit d'une propriété externe au protocole, un artefact qui a été conservé parce qu'il avait son utilité, même s'il avait aussi ses inconvénients. On peut noter d'ailleurs que le positionnement du statut est de plus une spécificité de `Test::More`, les scripts produisant leur sortie `ok / not ok` à la main ne le positionnant typiquement pas (sauf quand ils meurent violemment), pas plus que certains scripts basés sur `Test`, qui peuvent fournir un statut de succès malgré l'échec de certains points de tests.

En résumé, cette contrainte, si elle en est une pour le programmeur, permet aussi de mieux limiter le périmètre d'exécution du script et donc de mieux diagnostiquer les problèmes. Pour compenser, nous allons vous montrer quelques techniques pour aider le comptage des tests.

Techniques de comptages

Un premier moyen simple est d'utiliser `no_plan`...

Oui, on vient de vous dire qu'il ne fallait pas l'utiliser, mais calmez-vous et écoutez plutôt la suite de l'explication `:-)`

Un premier moyen simple, donc, est d'utiliser `no_plan` pendant la phase d'écriture du script, puis, une fois que celui-ci est en grande partie écrit, il suffit de l'exécuter de manière verbeuse avec `prove -bv t/script.t` par exemple, et de relever le nombre total de tests exécutés. Il suffit alors de positionner votre plan à cette valeur. Par la suite, il suffira de l'incrémenter lors de l'ajout d'autres points de tests.

Pour les scripts de tests s'appuyant sur des données dont la quantité peut varier ou être paramétrée, une bonne technique consiste justement à déterminer le nombre de tests par donnée analysée. Il suffit ensuite de calculer le nombre total de tests par une simple multiplication (et éventuellement l'addition des tests non dépendants des données). Ainsi, la plupart des scripts de test du module `Net::Pcap` commencent de la manière suivante :

```
my $total = 10;  # number of packets to process
plan tests => $total * 19 + 5;
```

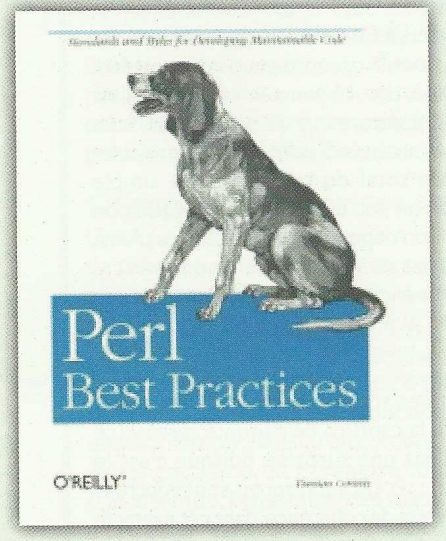
Plusieurs des scripts sont en effet orientés autour de la capture de paquets réseau (ce qui n'est pas une surprise, puisque c'est le rôle de `Net::Pcap`), sur lesquels sont effectués plusieurs tests de cohérence, dans cet exemple 19 par paquet. Si les tests portent sur des données spécifiques, il suffit de le stocker dans un endroit facilement accessible et de compter la quantité de données avant la déclaration du plan. « L'endroit » peut être par exemple une chaîne, un tableau, un hash ou dans la partie `DATA` du script. Une

bonne illustration de cette méthode est donnée dans les scripts du module `Pod::POM::View::HTML::Filter` :

```
    my @tests = map { [ split /^---.*?^/ms ] }
split /^===.*?^/ms, << 'TESTS';
    =begin filter foo
    bar foo bar
    baz
    =end
    ---
bar bar bar
    baz
    ===
    ...
    ===
    =begin filter verb
        verbatim block
    verbatim textblock
    =end
    ---
    verbatim block
verbatim textblock
    TESTS
    plan tests => scalar @tests + 2;
```

Ici, les données sont stockées sous forme d'enregistrements séparés par des `===`, chaque enregistrement étant lui-même constitué de deux parties séparées par des `---`. La première partie correspond à des fragments de documents Pod, la seconde au résultat attendu de la conversion du fragment par le module testé, `Pod::POM::View::HTML::Filter` **[1]**.

Perl Best Practices



Comme le titre l'indique, Damian Conway a rassemblé dans ce livre 256 règles de bonnes pratiques d'écriture de code Perl, basées sur sa propre expérience ainsi que de celle d'autres grands programmeurs Perl. Ces règles permettent, si on les suit, d'écrire un code qui sera lisible pour un maximum de personnes, facilitant ainsi d'autant sa maintenabilité. À peu près tous les sujets qu'un programmeur Perl pourrait rencontrer sont traités, des boucles et structures de contrôles aux hiérarchies de classes et d'objets, en passant par la gestion des entrées-sorties, des erreurs, des options en ligne de commande, ainsi que les meilleures manières d'écrire et d'utiliser les fonctions, les expressions régulières et la documentation.

Il propose aussi un chapitre consacré aux tests, conseillant de ne pas hésiter à utiliser `Test::More` et consorts dès que possible, par exemple pour transformer en script de test tout bug applicatif rapporté.

Écrivant avec son humour habituel, Damian explique en détail le pourquoi de chaque règle. Ainsi, même si on n'est pas de prime abord d'accord avec ses choix, il est très intéressant de lire ses explications, qui décrivent en quoi cela peut sinon nuire à la lisibilité et à la compréhension du code.

On ne peut résister à l'envie de vous citer la première et probablement l'une des plus importantes règles données par Damian :

> *Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.*

Disponible en français sous le titre *De l'art de programmer en Perl.*

Un autre exemple de cette technique est donné dans le script de test principal du module `File::Read` :

```
my @one_result_tests = (
    ...
    {
        args     => [ File::Spec->catfile(qw(t samples pi)) ],
        expected => "3.14159265358979\n",
    },
    {
        args     => [ File::Spec->catfile(qw(t samples hello)) ],
        expected => 'Hello',
    },
    {
        args     => [ File::Spec->catfile(qw(t samples world)) ],
        expected => "world\n",
    },
    ...
);
```

Le concept est exactement le même, sauf que les données sources et les résultats attendus sont stockés dans une structure Perl, réduisant ainsi le comptage des tests à quelques multiplications et additions triviales :

```
    plan tests => 5 + @one_result_tests * 2 + ...
```

Enfin, une dernière technique, utilisée par Jérôme Quelin, permet de simplifier encore le comptage. Elle consiste à utiliser des blocs `BEGIN` pour effectuer le compte, bloc par bloc, au fur et à mesure de la compilation du script :

```
    use strict;
    use Test::More;
    my $tests;              # nombre total de tests
    plan tests => $tests;   # déclaration du plan
    BEGIN { $tests += 4 }   # tests du chargement et de l'API
    use_ok('WebBrowser');
    can_ok('WebBrowser', qw(new clone load save));
    my $browser = WebBrowser->new
    isa_ok($browser, 'WebBrowser');
```

```
can_ok($browser, qw(clone load save));
BEGIN { $tests += 5 }   # tests de la méthode load()
my $test_page = 'test.html';
eval { $browser->load($test_page) };
is( $@, '', "méthode load(), argument : $test_page" );
ok( $browser->status_ok, "page correctement chargée" );
is( $browser->current_page->location, $test_page );
is( $browser->current_page->size, -s $test_page );
like( $browser->current_page->title, '/Page de test/' );
```

L'astuce est que les blocs `BEGIN` sont exécutés dès la phase de compilation du script. Comme la variable `$tests` a tout de même été déclarée au début du script (3e ligne), elle est déjà allouée et connue. Les blocs `BEGIN` incrémentent ensuite sa valeur au fur et à mesure de la compilation du script. Lorsque celui-ci est finalement exécuté, la variable `$tests` contient donc le nombre total de tests, qui est donné comme plan.

Écriture par paragraphe

Recommandée par Damian Conway dans *Perl Best Pratices* **[3]**, ce mode d'écriture consiste à séparer le code en petits ensembles de lignes, plus faciles à digérer qu'un gros bloc.

Cette technique est particulièrement bien adaptée pour les scripts de tests nécessitant d'être un peu long. Elle se marie de plus à merveille avec l'écriture par paragraphe et permet ainsi de simplifier d'autant la maintenance du programme : le compte des tests effectués se faisant localement à chaque paragraphe, cela limite les possibilités de se tromper.
À noter que le conseil de la section précédente de bien compartimenter les tests dans des scripts séparés prend maintenant tout son sens. L'autre avantage de la compartimentation est que cela limite le nombre de tests par script, facilitant par la-même leur décompte, quelle que soit la technique retenue.

Intégration au cœur du processus

Distribution CPAN

Petit rappel : quand on parle de « distribution CPAN », cela signifie qu'on parle d'une archive qui suit la structure et le processus d'installation classique (par `Makefile.PL` ou `Build.PL`), mais cela n'implique pas nécessairement de déposer cette archive sur le CPAN.

Ce qui rend l'écriture de tests si facile en Perl est la remarquable intégration des phases de test dans les cycles de développement et de déploiement des distributions CPAN, qui représentent le mécanisme d'installation le plus adapté pour les modules et programmes Perl.

Si l'exécution des tests lors de la phase de déploiement peut sembler redondante, il faut simplement se rendre compte qu'elle répond à une problématique différente. En effet, l'exécution des tests pendant la phase de développement permet à la fois la validation du code écrit et un débogage facilité par la vérification constante de non-régression, comme cela a été expliqué dans le précédent article. Lors de la phase de déploiement du logiciel, les tests permettent de vérifier et de valider le fonctionnement de tout ou partie de ces fonctionnalités en conditions de production. Car, qu'il s'agisse de code fonctionnel ou de la suite de tests, il s'agit d'être sûr que tout cela fonctionne ailleurs que sur le poste du développeur !

Phase de développement

Pendant le développement d'un module ou d'un programme Perl, plusieurs moyens existent pour exécuter les scripts de tests. Le plus classique est d'utiliser les commandes `make test` ou `Build test` qui effectuent les étapes de compilation, si nécessaire, et de lancement de la suite de tests. Lors de la compilation, les fichiers sources sont copiés dans le sous-répertoire `blib/`, les fichiers Perl étant placés dans l'arborescence `blib/lib/` et les fichiers binaires dans `blib/arch/`. La cible `test` exécute ensuite les scripts de tests au travers de `Test::Harness` pour afficher le résumé habituel :

```
$ ./Build test
lib/Module/ThirdParty.pm -> blib/lib/Module/ThirdParty.pm
t/00load.....ok 1/1# Testing Module::ThirdParty 0.16, Perl 5.008005,
/usr/bin/perl5.8.5
t/00load.....ok
t/00prereq....# Checking required modules
t/00prereq....ok
t/01api.......ok
t/10run.......ok
t/distchk.....ok
                    1/10 skipped: Module::Build PREREQ_PM not yet
implemented
t/pod.........ok
t/podcover....ok
t/portfs......ok
All tests successful, 1 subtest skipped.
Files=8, Tests=43,  1 wallclock secs ( 1.12 cusr +  0.14 csys =  1.26
CPU)
```

C'est bien, mais si la suite de tests est longue, on perd du temps pour rien à exécuter toute la suite quand on sait que l'on ne travaille que sur une partie bien déterminée, et vérifiée par un script particulier.
Il suffit donc de n'exécuter que ce script spécifique, mais en pensant à ajouter, au chemin de recherche, les arborescences dans `blib/` pour que les modules sur lesquels on travaille soient trouvés :

```
$ perl -wT -Iblib/lib -Iblib/arch t/10run.t
1..25
ok 1 - SVN::Core is a known third-party module
ok 2 - CAIDA::NetGeoClient is a known third-party module
ok 3 - Text::ChaSen is a known third-party module
...
```

Pour simplifier cette utilisation, on peut utiliser le module `blib` (intégré à la distribution standard de Perl) qui réalise cela de manière plus complète. Il a toutefois le défaut de ne pas être compatible avec le mode teinté, d'où l'intérêt de savoir aussi comment faire sans lui.

```
$ perl -w -Mblib t/10run.t
1..25
ok 1 - SVN::Core is a known third-party module
ok 2 - CAIDA::NetGeoClient is a known third-party module
ok 3 - Text::ChaSen is a known third-party module
...
```

L'inconvénient dans l'utilisation de `blib/` est qu'il faut par contre penser à exécuter `make` ou `Build` avant de lancer le script, afin de mettre à jour les fichiers dans ce répertoire. Pour les modules « pur Perl », sans partie à compiler comme du C ou de l'XS, on peut simplifier le processus en incluant directement l'arborescence contenant les fichiers sources de travail (si vous avez bien rangé vos modules dans `lib/`, ce que vous devriez faire de toute façon) :

```
$ perl -wT -Ilib t/10run.t
1..25
ok 1 - SVN::Core is a known third-party module
ok 2 - CAIDA::NetGeoClient is a known third-party module
ok 3 - Text::ChaSen is a known third-party module
...
```

C'est pas mal, mais un dernier problème est que l'on voit chaque point de test individuellement, sans le résumé bien pratique fourni par `Test::Harness`. Si le script comporte beaucoup de points de tests, ça défile à toute vitesse et on n'a pas le temps de voir les erreurs. La solution est de passer par `Test::Harness`, mais son appel sur la ligne de commande n'est pas des plus courts. Pour pallier cela, Andy Lester fournit avec les versions récentes de `Test::Harness` l'utilitaire **prove** qui permet justement d'exécuter un ou plusieurs scripts de test au travers de ce module.

Ses options les plus intéressantes sont `-b` et `-l` qui ajoutent respectivement `blib/lib/` et `lib/` au chemin de recherche, `-I` pour ajouter des chemins arbitraires, comme l'option de **perl**, `-t` et `-T` pour activer le mode teinté en mode d'avertissement ou en mode complet, `-r` pour chercher les scripts récursivement.

```
$ prove -l t/10run.t
t/10run....ok
All tests successful.
Files=1, Tests=25,  0 wallclock secs ( 0.06
cusr +  0.00 csys =  0.06 CPU)
```

Ainsi, si une erreur survient, elle devient bien plus facile à détecter :

```
$ prove -l t/10run.t
t/10run....NOK 19
#   Failed test ' - checking name'
#   in t/10run.t at line 33.
#          got: 'Chassen'
#     expected: 'ChaSen'
# Looks like you failed 1 test of 25.
t/10run....dubious
        Test returned status 1 (wstat 256, 0x100)
DIED. FAILED test 19
        Failed 1/25 tests, 96.00% okay
Failed Test Stat Wstat Total Fail  Failed  List of Failed
-------------------------------------------------------
t/10run.t      1   256    25    1   4.00%  19
Failed 1/1 test scripts, 0.00% okay. 1/25 subtests failed,
96.00% okay.
```

Comme précédemment, si on utilise le répertoire `blib/`, il faut penser à exécuter `make` ou `Build` juste avant.

Phase de déploiement

La phase de déploiement correspond à l'installation du module ou du programme concerné. Celle-ci peut se réaliser manuellement par l'exécution des commandes habituelles :

```
$ perl Makefile.PL && make test && sudo make install
```

dans le cas d'une distribution utilisant `ExtUtils::MakeMaker` ou

```
$ perl Build.PL && ./Build test && sudo ./Build install
```

si elle utilise `Module::Build`. Toutefois, il est largement préférable de passer par l'un des outils interactifs standards d'installation tel que le *shell* `CPAN.pm` ou par CPANPLUS. Ceux-ci offrent l'avantage d'exécuter ces commandes, en ajoutant éventuellement des paramètres supplémentaires pour tenir compte des spécificités de votre installation (par exemple, l'ajout d'un `PREFIX`). Mais le principal intérêt de ces shells est qu'ils se chargent de résoudre et d'installer automatiquement les dépendances du logiciel demandé. Il suffit de lancer l'installation et de regarder l'écran défiler `:-)`

Pour l'installation de chaque module, le shell exécute la suite de tests et ne réalise l'installation que si ces derniers ont réussi. Le principe est de vérifier et de valider que l'ensemble des composants fonctionnent comme prévu en conditions de production. La suite de tests offre donc un certain degré de garantie de bon fonctionnement des composants et, par la même, du logiciel demandé. Ce degré est bien évidemment dépendant de la qualité de la suite de tests, un point qui sera présenté ultérieurement.

Avec CPANPLUS, il est même possible d'envoyer des rapports pour indiquer si la suite de tests s'est correctement exécutée ou pas. Ces mails sont collectés et analysés dans le cadre du projet CPAN Testers **[5]**. Les pages de ce site sont pointées par les sites comme Search CPAN et Kobes' CPAN Search depuis les pages correspondant aux distributions. Cela permet ainsi d'évaluer la fiabilité et la portabilité d'un logiciel en vérifiant si sa suite de test réussit ou pas sur les systèmes des contributeurs à CPAN Testers.

Lorsque la suite de tests échoue lors du déploiement, il s'agit très souvent des mêmes quelques causes classiques : par exemple, du fait de l'absence d'un fichier nécessaire pour les tests, que le développeur a oublié de lister dans `MANIFEST`, et qui n'est donc pas inclus dans la distribution. Pour détecter au plus tôt ce type de problème, le développeur doit prendre l'habitude d'exécuter la cible `disttest`, qui réalise en partie une simulation des tests en phase de déploiement. Plus exactement, `disttest` crée la distribution dans sa forme finale, et se place dedans pour lancer la compilation et les tests. Les problèmes comme les fichiers manquants ou mal nommés dans `MANIFEST`, ou encore les erreurs de chemins relatifs sont ainsi immédiatement mis en évidence.

Un autre problème classique est d'oublier d'ajouter un module à la liste des pré-requis. L'aspect ennuyeux est que cela ne peut se détecter que quand on installe le logiciel sur une distribution Perl où ledit pré-requis est absent. La solution pour le développeur est d'essayer de déployer son logiciel sur une installation neuve de Perl, afin de vérifier que le processus se déroule sans accroc. Cela implique donc qu'il doit recréer cette installation chaque fois qu'il veut tester son logiciel, ce qui est clairement fastidieux. Il est bien sûr possible d'automatiser cela, et c'est l'un des buts du projet PITA **[6]**, initié par Adam Kennedy. Ce projet, assez ambitieux,

a pour but de fournir une infrastructure logicielle permettant de tester un logiciel sur des systèmes et plates-formes variés en utilisant les fonctionnalités d'émulateurs tels que Qemu et VMware. Encore à ses débuts, PITA est néanmoins une future solution très prometteuse.

Sélection des tests

Un point important à noter est que tous les tests ne sont pas forcément adaptés ni même souhaitables en phase de déploiement, par exemple parce qu'ils vérifient des aspects secondaires comme la documentation ou la présence de fichiers de méta-données. Ils peuvent aussi nécessiter un environnement ou des outils spéciaux, qu'il n'y a pas de raison de mettre en place en production. Ces tests, s'ils sont utiles pour le développeur, ne le sont pas à l'installation du logiciel et n'ont donc pas à être exécutés lors de cette phase.

Il faut donc opérer une sélection des tests à exécuter en phase de déploiement, et disposer d'un moyen pour automatiser cette sélection. On distingue principalement trois méthodes pour ce faire.

La **non-inclusion des scripts de test** consiste tout simplement à ne pas les distribuer dans l'archive finale (il suffit de les retirer du fichier `MANIFEST`). Les scripts ne sont présents que sur la machine du développeur. C'est une solution adoptée par certains contributeurs du CPAN qui considèrent ainsi qu'ils évitent de participer à sa « pollution ». Toutefois, cela présente le défaut que si quelqu'un veut reprendre la maintenance d'un module, il ne dispose pas des scripts additionnels et il doit alors les demander au mainteneur précédent, quand c'est possible (cela peut être impossible pour des raisons pratiques, telle que la perte du disque dur, voire pour des causes plus graves comme la maladie ou le décès de la personne concernée). Pour cette raison, beaucoup d'autres développeurs préfèrent donc tous les inclure, mais les désactiver en utilisant l'un des mécanismes suivants.

Le premier est **l'exécution conditionnelle à la présence du module de test**, c'est-à-dire que le test n'est exécuté que si le module nécessaire est présent. Cela a l'avantage d'éviter de devoir l'ajouter à la liste des pré-requis du paquet alors qu'il n'est utile que pour les tests. La manière propre de gérer ça est d'utiliser `skip_all` :

```
use strict;
use Test::More;
eval "use Test::Pod";
plan skip_all => "Test::Pod required for testing POD" if $@;
# reste du script
```

On déplace le `use` du module dans un `eval`, ce qui permet de capturer proprement toute erreur de chargement. S'il y en a une, on utilise l'option `skip_all` de la fonction `plan()` pour signaler que ce script doit être sauté. Sinon le reste du script est exécuté.

Le second est **l'exécution conditionnelle à un facteur externe**. Ce dernier peut être la présence ou l'absence d'un fichier, d'un répertoire d'une variable d'environnement, ou encore d'une clé de registre sous Win32. Là aussi, on utilise `skip_all` pour sauter le test. Par exemple, pour n'exécuter un test que si une variable d'environnement est positionnée :

```
plan skip_all => "set TEST_POD to enable this test" unless $ENV{TEST_POD};
```

Autre exemple, tiré des scripts de `Net::Pcap` :

```
use strict;
use Test::More;
use lib 't';
use Utils;
plan skip_all => "must be run as root" unless is_allowed_to_use_pcap();
plan skip_all => "no network device available" unless find_network_device();
```

Ce script commence par ajouter le répertoire `t/` dans le chemin de recherche, puis charge le module `Utils`, qui est justement dans ce répertoire. Celui-ci fournit quelques fonctions destinées à faciliter l'écriture des tests de `Net::Pcap`. La première utilisée, `is_allowed_to_use_pcap()`, vérifie que l'utilisateur qui exécute le script a le droit d'ouvrir l'interface réseau en mode direct ; la seconde, `find_network_device()`, recherche une interface réseau utilisable pour les tests. Ces deux vérifications permettent de sauter le script si les conditions nécessaires à son exécution ne sont pas remplies et permettent ainsi d'éviter des erreurs inutiles.

Conclusion

Maintenant que toutes les bases sur les tests en Perl vous ont été présentées et expliquées, nous continuerons cette série d'articles en explorant les modules de tests disponibles sur le CPAN pour ajouter des sémantiques de plus haut niveau.

Sébastien Aperghis-Tramoni & Philippe Blayo,

« Maddingue » – {Marseille,Sophia}.pm

Sébastien Aperghis-Tramoni est administrateur systèmes dans le Sud de la France. Il maintient par ailleurs une vingtaine de modules sur le CPAN et essaye d'aider au développement de Perl en fournissant des services comme *Perl cover* et un Gonzui du CPAN.

Philippe Blayo – Paris.pm

RÉFÉRENCES

- **[1]** À noter qu'il s'agit du module écrit par Philippe Bruhat et qui est utilisé pour générer les articles des Mongueurs sur le site Articles.
- **[2]** APERGHIS-TRAMONI (Sébastien), « Créer une distribution pour le CPAN », *GNU/Linux Magazine France* N° 69, http://articles.mongueurs.net/magazines/linuxmag69.html
- **[3]** CONWAY (Damian), *Perl Best Practices*, O'Reilly & Associates, 2005, ISBN 0-596-00173-8, http://www.oreilly.com/catalog/perlbp/. Disponible en français sous le titre *De l'art de programmer en Perl*, O'Reilly 2006, ISBN 2-84177-369-8.
- **[4]** QA Podcasts – http://www.qapodcast.com/
- **[5]** CPAN Testers – http://testers.cpan.org/
- **[6]** PITA – http://ali.as/pita/, http://search.cpan.org/dist/PITA/

Merci aux Mongueurs qui ont assuré la relecture de cet article.

Programmation noyau sous Linux
Pilotes en mode caractère

Pierre Ficheux

EN DEUX MOTS **Ce nouvel article de la série est consacré à la programmation des pilotes de périphériques en mode dit « caractère ». Les connaissances acquises lors du premier article consacré à l'API des modules Linux vont nous permettre d'aborder assez simplement la notion de « pilote Linux » qui, finalement, est un module dans une version un peu plus ardue.**

Nous aborderons les sujets suivants :

- l'introduction aux pilotes Linux : les différents types de pilotes ;
- l'interface avec l'espace utilisateur : le répertoire `/dev` ;
- l'API des pilotes en mode caractère ;
- le transfert de données avec l'espace utilisateur ;
- l'allocation dynamique de mémoire ;
- la méthode `ioctl` ;
- les files d'attente.

Les exemples fournis sont inspirés de ceux développés par Stelian Pop pour la formation « noyau Linux » d'Open Wide.

Introduction aux pilotes Linux

Le système Linux étant de la famille UNIX, la structure d'un pilote Linux est relativement proche de celle d'un pilote générique UNIX. Un pilote est une portion de code exécutée dans l'espace du noyau. Il est chargé de faire l'interface entre un programme utilisateur et un composant matériel. Ce dernier point n'est pas forcément vrai puisqu'il existe des pilotes de périphériques *virtuels* tels les systèmes de fichier. En toute rigueur, un programme UNIX devrait systématiquement accéder à un périphérique au travers d'un pilote. Ce n'est pas toujours le cas, puisque nous avons vu dans plusieurs articles précédents qu'il était possible – sous Linux – d'accéder à une ressource matérielle grâce à la fonction `ioperm()`, puis aux fonctions `inb()` et `outb()`. Cette méthode est cependant peu recommandée et rarement utilisée sauf pour des cas simples ou très particuliers.

L'API d'un pilote Linux est très fortement inspirée de celle des pilotes UNIX des années 1970. Nous rappelons qu'elle est basée sur l'utilisation de fonctions ou méthodes permettant d'effectuer des actions basiques.

- une méthode `open()` permettant d'ouvrir le périphérique ;
- une méthode `close()` permettant de fermer (libérer) le périphérique. Sous Linux, cette méthode est nommée `release()` ;
- une méthode `read()` permettant de lire des données du périphérique ;
- une méthode `write()` permettant d'écrire des données sur le périphérique ;
- une méthode `ioctl()` (*Input Output ConTroL*) permettant de configurer le périphérique.

Il existe d'autres méthodes comme `lseek()`, `poll()` ou `mmap()`, mais elles sont moins fréquemment utilisées dans les pilotes simples.

Les méthodes sont activées depuis un programme de l'espace utilisateur en passant par les fichiers spéciaux du répertoire `/dev`. Ces fichiers sont appelés nœuds (*nodes* ou *devices* en anglais). Nous décrirons plus précisément ce répertoire plus loin dans le document.

Le pilote est conforme à l'API des modules Linux décrite dans l'article précédent. Du point de vue fonctionnel, l'API des modules n'a rien à voir avec les pilotes. Elle permet simplement de charger *dynamiquement* le pilote dans l'espace du noyau en cours d'exécution.

REMARQUE

Un pilote externe aux sources du noyau Linux sera *toujours* développé sous forme de module.

Il existe différents types de pilotes, liés aux types de périphériques qu'ils peuvent contrôler.

- Les pilote en mode *caractère (char device driver)*. Ces pilotes sont destinés à manipuler les périphériques les plus courants avec lesquels ils échangent des données sous forme de flux d'octets de taille variable (minimum 1 octet). De ce fait, la plupart des pilotes de périphérique sous Linux seront en mode caractère. L'exemple le plus courant est le pilote de l'interface série RS-232.
- Les pilotes en mode *bloc (block device driver)*. Ces pilotes sont destinés à manipuler des périphériques de stockage avec lesquels ils échangent des blocs de données (disque dur, CDROM, DVD, disque mémoire, etc.). La taille des blocs peut être de 512, 1024, 2048 (ou plus) octets suivant le périphérique.
- Les pilotes de périphériques réseau (*network device driver*). Ils sont destinés à gérer des contrôleurs réseau (exemple : carte Ethernet), mais aussi des piles de protocoles. Contrairement aux autres pilotes, ils n'utilisent pas d'entrée dans le répertoire `/dev`.

A cette liste, nous pouvons également ajouter quelques pilotes spéciaux le plus souvent dédiés à des bus ou des API particulières, citons entre autres :

- le bus PCI ;
- le bus USB ;
- les pilotes vidéo (V4L et V4L2).

La structure générale du noyau Linux est présentée dans la figure ci-dessous. On peut y voir le positionnement des principaux types de pilotes (schéma par Julien Gaulmin).

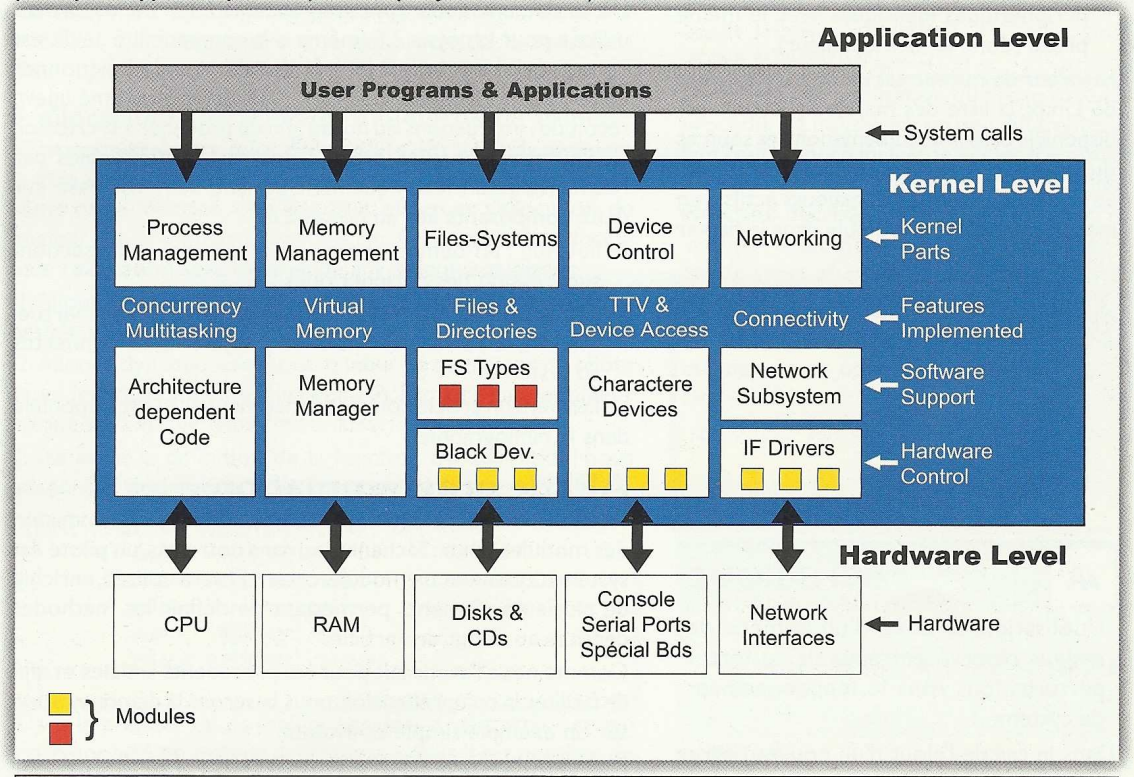


Figure 1 : Structure du noyau Linux

Dans le présent article, nous traiterons uniquement le cas des pilotes en mode caractère.

Quelques règles d'usage

La programmation d'un pilote est notoirement plus complexe que celle d'un programme en espace utilisateur.

- Le noyau est un élément fondamental du système et un pilote mal conçu peut entraîner des dysfonctionnements importants, voire un arrêt du système, ce qui n'est pas le cas dans l'espace utilisateur.
- La mise au point dans l'espace noyau est complexe (voir l'article « Débogage dans l'espace noyau avec KGDB » dans le magazine LM 88).
- Les fonctions de la `glibc` ne sont pas disponibles dans l'espace noyau, mais certaines sont implémentées dans le répertoire `lib` des sources du noyau.
- Un pilote doit être programmé en C (pas de C++), mais la structure d'un pilote est « orientée objet ».
- En toute rigueur, on doit respecter le style de programmation défini dans le fichier `Documentation/CodingStyle` des sources du noyau.
- Un pilote doit prendre en compte les différentes architectures, particulièrement au niveau des « *big-endians* » et « *little-endians* ».

Tout cela doit inciter le lecteur à concevoir des pilotes les plus simples possibles et de reporter la difficulté sur l'espace utilisateur. De plus, nous rappelons une nouvelle fois qu'un pilote Linux doit en théorie être diffusé sous GPL, ce qui peut poser des problèmes par rapport à certains codes sensibles.

Le point de vue de l'espace utilisateur

Dans le cas d'un pilote en mode caractère, le périphérique est vu depuis un programme utilisateur comme un fichier spécial du répertoire `/dev`. Le fichier est dit « spécial », car il n'occupe quasiment pas d'espace sur le disque (uniquement le point d'entrée). Le but du fichier spécial est uniquement de faire un lien entre l'espace utilisateur et le pilote de périphérique associé.

Principe du majeur/mineur

Si l'on regarde le fichier spécial associé au premier port série, on obtient :

```
$ ls -l /dev/ttyS0
crw-rw---- 1 root uucp 4, 64 oct 31 10:34
/dev/ttyS0
```

Le premier caractère identifie le *type* de périphérique, soit ici la lettre *c* pour caractère. Les neufs caractères suivants correspondent aux droits d'accès habituels sous Linux. Il en est de même pour le propriétaire et le groupe.

Par contre, les deux champs qui suivent sont particuliers aux fichiers spéciaux.

- La première valeur appelée « majeur » identifie le type de périphérique (ici le port série).
- La seconde valeur appelée « mineur » identifie l'instance du périphérique. Ces mineurs permettent de gérer plusieurs périphériques identiques avec le même pilote (donc le même majeur).

La valeur du majeur est unique. Dans le cas de Linux, la liste des majeurs réservés est disponible dans la documentation des sources du noyau, soit le fichier `Documentation/devices.txt`. La liste des majeurs actifs à un instant donné est disponible dans le fichier `/proc/devices`.

```
$ cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
...
```

REMARQUE

L'utilisation erronée d'un numéro de majeur réservé entraînerait de fortes perturbations dans le fonctionnement du système !

Dans le cas de l'ajout d'un nouveau pilote par l'utilisateur, il sera préférable d'utiliser les fonctionnalités d'allocation dynamique de majeur ou de mineur plutôt que d'utiliser une valeur fixe. Ce point sera abordé dans la description de l'API.

La création d'un nouveau fichier spécial s'effectue grâce à la commande `mknod`. Si l'on devait créer le fichier spécial `/dev/ttyS0`, on utiliserait la commande :

```
# mknod /dev/ttyS0 c 4 64
```

Pour supprimer un fichier spécial, on utilise la commande classique `rm`.

```
# rm -f /dev/ttyS0
```

Limites du système de majeur/mineur

Cette approche est simple, mais elle pose un problème de duplication d'information entre l'espace utilisateur et l'espace noyau. En effet, les fichiers spéciaux sont créés dans `/dev` avec des valeurs passées à `mknod` ou bien à des utilitaires chargés de créer une liste de fichiers spéciaux (exemple : `MAKEDEV`). Pour que le système fonctionne, il faut que ces mêmes valeurs soient utilisées dans le code source des pilotes.

L'approche `devfs` permettait d'améliorer les choses pour le noyau 2.4. Grâce à `devfs`, les périphériques ne sont plus manipulés sous forme de majeur/mineur, mais sous forme de nom. L'entrée dans `/dev` est créée dynamiquement lors de l'insertion du pilote. Cependant le code de `devfs` est situé dans le noyau ce qui pose des problèmes de nommage et de compatibilité avec le standard LSB (*Linux Standard Base*) qui ne traite pas l'espace noyau.

De ce fait, une autre approche appelée `udev` est désormais utilisée pour le noyau 2.6 (même si la compatibilité `devfs` est conservée). Le principal intérêt de `udev` est de fonctionner entièrement en espace utilisateur. Un démon nommé `udevd` reçoit des instructions du noyau afin de procéder à la création de l'entrée dans `/dev` grâce à des règles modifiables par l'administrateur. Techniquement parlant, `udev` est basé sur deux composants liés au noyau 2.6.

- `hotplug` : un démon chargé de la gestion de l'insertion/suppression des périphériques ;
- `sysfs` (monté sur `/sys`) : un système de fichier virtuel similaire à `/proc` décrivant les périphériques connectés au système.

Le lien vers un article complet concernant `udev` est disponible dans la bibliographie.

API de programmation

Nous avons vu, dans l'article précédent, l'API de programmation des modules Linux. Sachant que, dans notre cas, un pilote est systématiquement un module, cette API sera utilisée, enrichie de plusieurs éléments permettant de définir les méthodes décrites au début de l'article.

Comme nous l'avons fait pour les précédents articles et afin de faciliter la compréhension, nous baserons la démonstration sur un exemple simple et évolutif.

La structure file_operations

Cette structure dont le type est décrit dans le fichier `<linux/fs.h>` permet de définir les méthodes du pilote. La tradition veut que l'on préfixe le nom de la structure et des méthodes par le nom du pilote (ici `mydriver`).

```
static struct file_operations mydriver_fops = {
                    .owner =    THIS_MODULE,
                    .read =     mydriver_read,
                    .write =    mydriver_write,
                    .open =     mydriver_open,
                    .release =  mydriver_release,
};
```

De part la nécessité de définir les symboles correspondant aux méthodes, on placera la déclaration de la structure *après* la définition des méthodes. De ce fait, l'architecture générale du pilote sera la suivante :

1. Déclaration des en-têtes.
2. Déclaration des macro-instructions identifiant le pilote.
3. Déclaration des variables.
4. Définition des méthodes.
5. Définition de la structure `file_operations`.
6. Définition des points d'entrée `module_init()` et `module_exit()` du module.

Déclarations à effectuer

Dans le cas simple qui nous intéresse, les déclarations sont similaires à celles que nous avons effectuées pour l'exemple des modules de l'article précédent.

```
#include <linux/kernel.h>       /* printk() */
```

```
#include <linux/module.h>       /* modules */
#include <linux/init.h>         /* module_{init,exit}() */
#include <linux/fs.h>          /* file_operations */
MODULE_DESCRIPTION("mydriver1");
MODULE_AUTHOR("Stelian Pop/Pierre Ficheux, Open Wide");
MODULE_LICENSE("GPL");
```

La seule nouveauté est la présence du fichier d'en-tête <linux/fs.h> nécessaire à la définition de la structure file_operations.

L'allocation/libération du majeur ou du mineur

Chronologiquement, la fonction module_init() est la première à être exécutée lors du chargement du module. C'est donc dans cette fonction que l'on doit effectuer l'allocation du majeur ou du mineur suivant les cas. Si l'on part du principe que l'allocation sera dynamique, il y a deux solutions :

1. Allouer dynamiquement un majeur. Ce majeur sera ensuite visible dans /proc/devices.

2. Allouer dynamiquement un mineur lié au majeur du pilote misc (correspondant à la valeur 10). Dans ce cas, le mineur alloué sera visible dans /proc/misc.

Juste avant la définition de la fonction d'initialisation, nous plaçons la déclaration de la structure file_operations.

```
static struct file_operations mydriver1_fops = {
          .owner =   THIS_MODULE,
          .read =    mydriver1_read,
          .write =   mydriver1_write,
          .open =    mydriver1_open,
          .release = mydriver1_release,
};
```

Dans le premier cas, le majeur par défaut vaut zéro – ce qui correspond à un majeur dynamique – mais il est possible de passer une valeur en paramètre du module, soit :

```
static int major = 0; /* Major number */
module_param(major, int, 0644);
MODULE_PARM_DESC(major, "Static major number (none = dynamic)");
```

L'enregistrement du pilote s'effectue grâce à la fonction register_chrdev(). Si le majeur vaut zéro, le système retournera un majeur dynamique. La définition de la fonction d'initialisation du module est la suivante :

```
static int __init mydriver1_init(void)
{
     int ret;
     ret = register_chrdev(major, "mydriver1", &mydriver1_fops);
     if (ret < 0) {
               printk(KERN_WARNING "mydriver1: unable to get a major\n");
               return ret;
     }
     if (major == 0)
          major = ret; /* dynamic value */
          printk(KERN_INFO "mydriver1: successfully loaded with major
          %d\n", major);
          return 0;
}
```

La libération du majeur alloué s'effectue dans la fonction de déchargement du module dont le code source est le suivant. Pour cela, on utilise la fonction unregister_chrdev().

```
static void __exit mydriver1_exit(void)
{
          if (unregister_chrdev(major, "mydriver1") < 0) {
               printk(KERN_WARNING "mydriver1: error while unregistering\n");
               return;
               }
          printk(KERN_INFO "mydriver1: successfully unloaded\n");
}
```

La fin du programme correspond simplement à l'API des modules Linux.

```
module_init(mydriver1_init);
module_exit(mydriver1_exit);
```

Dans le cas de l'utilisation du pilote misc, la procédure à utiliser est un peu différente. Il faut tout d'abord inclure le fichier d'en-tête <linux/miscdevice.h>, puis déclarer une structure décrivant le pilote.

```
#include <linux/miscdevice.h>   /* misc driver interface */
static struct miscdevice mymisc; /* Misc device handler */
```

Dans la fonction d'initialisation du module, on alloue dynamiquement le mineur au lieu du majeur en utilisant la fonction misc_register().

```
static int __init mydriver2_init(void)
{
  int ret;
  mymisc.minor = MISC_DYNAMIC_MINOR;
  mymisc.name = "mydriver2";
  mymisc.fops = &mydriver2_fops;
  ret = misc_register(&mymisc);
  if (ret < 0) {
    printk(KERN_WARNING "mydriver2: unable to get a dynamic minor\n");
    return ret;
  }
  return 0;
}
```

Pour libérer le mineur alloué, on utilise la fonction misc_deregister().

```
static void __exit mydriver2_exit(void)
{
  misc_deregister(&mymisc);
  printk(KERN_INFO "mydriver2: successfully unloaded\n");
}
```

Les méthodes open() et release()

Ces deux méthodes sont les points d'entrée principaux d'un pilote UNIX, même si, sous Linux, certaines tâches d'initialisation peuvent être reportées dans les fonctions module_init() et module_exit(). La méthode open() est réservée à des tâches d'initialisation matérielle du périphérique (exemple : initialisation matérielle d'un bus ou allocation dynamique de mémoire). La méthode release() effectue la libération des ressources allouées. Dans le cas de l'exemple, les méthodes effectuent uniquement un affichage.

```
static int mydriver1_open(struct inode *inode, struct file *file)
{
                    printk(KERN_INFO "mydriver1: open()\n");
                    return 0;
}
static int mydriver1_release(struct inode *inode, struct file *file)
{
```

```
                printk(KERN_INFO "mydriver1: release()\n");
                return 0;
}
```

Dans d'autres pilotes plus complexes, on peut entre autres tester la validité du mineur. Voici l'exemple du pilote du port parallèle.

```
static int lp_open(struct inode * inode, struct file * file)
{
                unsigned int minor = iminor(inode);
                if (minor >= LP_NO)
                return -ENXIO;
                ...
}
```

Les méthodes read() et write()

Ces méthodes sont, en général, les plus utilisées, car elles permettent d'échanger des données avec le périphérique grâce à un tampon `buf`. Dans l'exemple simple qui suit, le code se contente d'afficher un message. Nous verrons plus loin les fonctions à utiliser pour échanger des données entre l'espace utilisateur et l'espace noyau.

```
static ssize_t mydriver1_read(struct file *file, char *buf, size_t
count, loff_t *ppos)
{
        printk(KERN_INFO "mydriver1: read()\n");
        return count;
}
static ssize_t mydriver1_write(struct file *file, const char *buf,
size_t count, loff_t *ppos)
{
        printk(KERN_INFO "mydriver1: write()\n");
        return count;
}
```

Compilation et test du pilote

Le fichier `Makefile` est identique à celui utilisé pour les exemples de modules Linux.

```
KDIR= /lib/modules/$(shell uname -r)/build
obj-m := mydriver1.o
all:
        $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
install:
        $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules_install
        depmod -a
clean:
        rm -f *~
        $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) clean
```

On teste le premier module en effectuant la compilation, puis l'insertion du module grâce à `insmod`.

```
$ make
# insmod mydriver1.ko
# dmesg
mydriver1: successfully loaded with major 253
```

Le système a affecté dynamiquement le majeur 253 au nouveau pilote, cependant le fichier spécial dans `/dev` n'est pas créé par défaut. Pour ce faire, on doit utiliser la commande `mknod` pour créer le fichier associé que nous appellerons `mydriver1`.

```
# mknod /dev/mydriver1 c 253 0
```

En pratique, on peut utiliser un script simple permettant de charger le module et de créer le fichier spécial en lisant `/proc/devices`.

```
#!/bin/sh
if [ $# -eq 0 ]; then
    echo "Usage: $0 module_name"
    exit 1
fi
insmod ${1}.ko
X=`grep $1 /proc/devices`
if [ "$X" != "" ]; then
    set $X
    rm -f /dev/$2
    mknod /dev/$2 c $1 0
else
    echo "Module $1 not loaded !"
    exit 1
fi
```

A partir de là, le pilote est accessible depuis l'espace utilisateur. Pour tester le pilote, il n'est pas nécessaire d'écrire un programme, puis le langage *shell* fournit les moyens grâce aux opérateurs de redirection. On peut tester l'ouverture, puis la fermeture par :

```
# < /dev/mydriver1
```

Ce qui provoque l'affichage suivant dans les traces du noyau :

```
mydriver1: open()
mydriver1: release()
```

On teste l'écriture par :

```
# echo x > /dev/mydriver1
```

Ce qui provoque l'affichage suivant dans les traces du noyau :

```
mydriver1: open()
mydriver1: write()
mydriver1: release()
```

Pour tester la lecture, on peut utiliser la commande `dd` :

```
dd bs=1 count=1 < /dev/mydriver1
```

Ce qui provoque l'affichage suivant dans les traces du noyau :

```
mydriver1: open()
mydriver1: read()
mydriver1: release()
```

Dans le cas du pilote utilisant l'allocation dynamique du mineur, la procédure est la même. On insère le module.

```
# insmod mydriver2.ko
```

Le point d'entrée correspondant apparaît automatiquement dans `/proc/misc` et dans le répertoire `/dev`.

```
# cat /proc/misc
 62 mydriver2
 63 device-mapper
175 agpgart
144 nvram
228 hpet
135 rtc
231 snapshot
# ls -l /dev/mydriver2
crw------- 1 root root 10, 62 nov  5 13:59 /dev/mydriver2
```

Cette apparition « magique » dans `/dev` n'en est pas vraiment une et la création du fichier spécial est effectuée par `udev` déjà cité au début de l'article (si le démon `udevd` est activé).
Le fonctionnement de `udev` est basé sur le système de fichier `sysfs` monté sur `/sys`. Le fichier spécial est créé, car un pilote de type `misc` possède une entrée dans le répertoire `/sys/class/misc`. Cette entrée est présente grâce à l'appel à `class_device_create()` présent dans la fonction `misc_register()`.

L'absence de classe dans le cas de l'allocation dynamique du majeur (voir l'exemple précédent, soit mydriver1) empêche la création automatique du fichier spécial.
Si nous revenons à l'exemple mydriver2, nous obtenons :

```
# ls -l /sys/class/misc
total 0
drwxr-xr-x 2 root root 0 nov  5 12:32 agpgart
drwxr-xr-x 2 root root 0 nov  5 12:32 device-mapper
drwxr-xr-x 2 root root 0 nov  5 12:32 hpet
drwxr-xr-x 2 root root 0 nov  5 14:02 mydriver2
drwxr-xr-x 2 root root 0 nov  5 12:32 nvram
drwxr-xr-x 2 root root 0 nov  5 12:32 rtc
drwxr-xr-x 2 root root 0 nov  5 12:32 snapshot
```

On peut augmenter le niveau de trace du démon udevd en modifiant le fichier /etc/udev/udev.conf.

```
# udev.conf
# The initial syslog(3) priority: "err", "info", "debug" or its
# numerical equivalent. For runtime debugging, the daemons
internal
# state can be changed with: "udevcontrol log_priority=<value>".
#udev_log="err"
udev_log="debug"
```

Lors de l'insertion du module mydriver2.ko, on obtient les traces suivantes dans le fichier /var/log/messages :

```
Nov  5 13:59:48 dhcp-588-2 udevd[438]: udev_event_run: seq 792
forked, pid [5983], 'add' 'module', 0 seconds old
Nov  5 13:59:48 dhcp-588-2 udevd[438]: udev_event_run: seq 793
forked, pid [5984], 'add' 'misc', 0 seconds old
Nov  5 13:59:48 dhcp-588-2 udevd-event[5983]: udev_rules_get_run:
rule applied, 'mydriver2' is ignored
Nov  5 13:59:48 dhcp-588-2 udevd-event[5983]: udev_device_event:
device event will be ignored
Nov  5 13:59:48 dhcp-588-2 udevd-event[5983]: udev_event_run: seq
792 finished
Nov  5 13:59:48 dhcp-588-2 udevd[438]: udev_done: seq 792, pid
[5983] exit with 0, 0 seconds old
Nov  5 13:59:48 dhcp-588-2 udevd-event[5984]: udev_rules_get_name:
no node name set, will use kernel name 'mydriver2'
Nov  5 13:59:48 dhcp-588-2 udevd-event[5984]: create_node:
creating device node '/dev/mydriver2', major = '10', minor = '62',
mode = '0600', uid = '0', gid = '0'
```

L'utilisation du pilote misc est donc intéressante, car elle évite l'étape de création du point d'entrée correspondant au majeur dynamique dont la valeur peut changer suivant la configuration du système. Le point d'entrée disparaît lorsque l'on décharge le module. Par contre, chaque pilote misc ne peut utiliser qu'un seul mineur.

```
# rmmod mydriver2
# ls -l /dev/mydriver2
ls: /dev/mydriver2: Aucun fichier ou répertoire de ce type
# cat /proc/misc
 63 device-mapper
175 agpgart
144 nvram
228 hpet
135 rtc
231 snapshot
```

Echange de données avec le pilote

Les programmes utilisateurs et les pilotes ne fonctionnant pas dans le même espace de mémoire, il est nécessaire d'utiliser des fonctions dédiées pour écrire ou lire des données vers ou en provenance du pilote. Les fonctions à utiliser sont déclarées dans <asm/uaccess.h>.

```
- copy_from_user(void *to, void *from, unsigned long size)
- copy_to_user(void *to, void *from, unsigned long size)
```

L'exemple que nous proposons utilise un tampon *statique* – un tableau de 64 caractères – permettant de recevoir des données par une écriture depuis l'espace utilisateur, puis de renvoyer ces mêmes données lors d'une lecture depuis cet espace utilisateur. Pour cela, nous devons déjà inclure le fichier <asm/uaccess.h>.

```
#include <asm/uaccess.h>        /* copy_{from,to}_user() */
```

Il faut ensuite déclarer le tampon, ainsi que le nombre d'éléments disponibles :

```
#define BUF_SIZE 64
static char buffer[BUF_SIZE];
static size_t num = 0; /* Number of available bytes in the buffer */
```

La méthode write() lit les données provenant de l'espace utilisateur dans la limite de la taille du tampon.

```
static ssize_t mydriver3_write(struct file *file, const char *buf,
size_t count, loff_t *ppos)
{
  size_t real;

  real = min((size_t)BUF_SIZE, count);

  if (real)
    if (copy_from_user(buffer, buf, real))
      return -EFAULT;

  num = real; /* Destructive write (overwrite previous data if any) */

  printk(KERN_DEBUG "mydriver3: wrote %d/%d chars %s\n", real,
count, buffer);

  return real;
}
```

La méthode read() retourne ces mêmes données à l'espace utilisateur et vide le tampon.

```
static ssize_t mydriver3_read(struct file *file, char *buf, size_t
count, loff_t *ppos)
{
  size_t real;

  real = min(num, count);

  if (real)
    if (copy_to_user(buf, buffer, real))
      return -EFAULT;

  num = 0; /* Destructive read (no more data after a read) */

  printk(KERN_DEBUG "mydriver3: read %d/%d chars %s\n", real, count,
buffer);

  return real;
}
```

Pour tester le pilote, nous utilisons la même méthode que précédemment. Tout d'abord, nous chargeons le module, puis nous écrivons une chaîne de caractères sur le fichier spécial associé.

```
# insmod mydrive3.ko
# echo -n salut > /dev/mydriver3
```

Ce qui provoque l'affichage suivant dans les traces du noyau :

```
mydriver3: open()
mydriver3: wrote 5/5 chars salut
mydriver3: release()
```

Nous pouvons ensuite lire le contenu du tampon stocké par le pilote.

```
# dd bs=5 count=1 < /dev/mydriver3 2> /dev/null
salut
#
```

Ce programme en C permet d'effectuer le même test :

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
main (int ac, char **av)
{
  char buf[64];
  int n, fd;
  fd = open (av[1], O_RDWR);
  if (fd < 0) {
    perror ("open");
    exit (1);
  }
  n = write (fd, av[2], strlen(av[2]));
  read (fd, buf, n);
  printf ("buf= %s\n");
  close (fd);
}
```

On utilise le programme comme suit :

```
# ./mydriver3_user /dev/mydriver3 salut
buf= salut
```

Allocation dynamique de mémoire

Dans l'espace utilisateur, l'allocation et la libération dynamique de mémoire s'effectuent grâce aux fonctions `malloc()` et `free()`. Ces fonctions ont leurs équivalents dans l'espace du noyau, en l'occurrence `kmalloc()` et `kfree()`. La taille allouée par la fonction `kmalloc()` est cependant limitée à 128 Ko de mémoire ce qui peut être insuffisant pour certains pilotes. Dans ce cas, il est possible d'utiliser des fonctions plus complexes comme `__get_free_page()`/`free_pages()` qui alloue/libère plusieurs pages de mémoire contiguës de mémoire *réelle* ou bien `vmalloc()`/`vfree()` qui alloue un espace continu de mémoire *virtuelle*.

Pour illustrer cela, nous allons modifier l'exemple précédent pour qu'il utilise un tampon de mémoire dynamique au lieu d'un tampon statique.

Il faut tout d'abord ajouter le fichier d'en-tête décrivant les fonctions.

```
#include <linux/slab.h>                    /* kmalloc()/kfree() */
```

La taille du tampon est par défaut de 64 octets, mais peut être modifiée en passant le paramètre au chargement du module.

```
static size_t buf_size = 64; /* Buffer size */
module_param(buf_size, int, 0644);
MODULE_PARM_DESC(buf_size, "Buffer size");
```

Le tampon est désormais un pointeur.

```
static char *buffer;              /* copy_from/to_user buffer */
```

L'allocation du tampon s'effectue dans la fonction `module_init()`. Le paramètre `GFP_KERNEL` indique une allocation normale de mémoire (pouvant « dormir »), alors que le paramètre `GFP_ATOMIC` indique une allocation « atomique », ce qui est nécessaire dans certains cas (exemple : une fonction de traitement d'interruption).

```
buffer = (char *)kmalloc(buf_size, GFP_KERNEL);
if (buffer != NULL) {
  printk(KERN_DEBUG "mydriver4: allocated a %d bytes buffer\n",
buf_size);
} else {
  printk(KERN_WARNING "mydriver4: unable to allocate a %d bytes
buffer\n", buf_size);
  misc_deregister(&mymisc);
  return -ENOMEM;
}
```

La libération de mémoire s'effectue dans la fonction `module_exit()`.

```
static void __exit mydriver4_exit(void)
{
  kfree (buffer);
  misc_deregister(&mymisc);
  printk(KERN_INFO "mydriver4: successfully unloaded\n");
}
```

La méthode ioctl()

Cette méthode est utilisée pour effectuer les opérations inaccessibles aux méthodes `read()` et `write()`. En général, cela correspond à des actions de configuration matérielle du périphérique (exemple : modifier la vitesse du port série). Il faut noter que la méthode `ioctl()` permet également de lire des valeurs à partir du pilote.

Dans notre exemple, nous considérons deux modes de fonctionnement (MODE1 et MODE2) affectés à la variable `my_mode` du pilote. La méthode `ioctl()` permet de lire et d'écrire les modes depuis l'espace utilisateur.

```
/* ioctl cmds */
#define SET_MODE 0
#define GET_MODE 1
/* ioctl valid args */
#define MODE1    1
#define MODE2    2
static int my_mode = MODE1;
```

Le code de la méthode `ioctl()` est le suivant. Il faut noter que la lecture du mode nécessite l'utilisation de la fonction `copy_to_user()` pour retourner la valeur à l'espace utilisateur. De même, l'utilisation de `copy_from_user()` sera nécessaire si le paramètre envoyé au pilote est passé par un pointeur de structure et non par un type de donnée simple.

```
static int mydriver5_ioctl(struct inode *inode, struct file *file,
                           unsigned int cmd, unsigned long arg)
{
  printk(KERN_DEBUG "mydriver5: ioctl()\n");
  switch (cmd) {
  case SET_MODE :
    switch (arg) {
    case MODE1 :
      my_mode = MODE1;
      break;
    case MODE2 :
      my_mode = MODE2;
      break;
    default :
```

```
      printk(KERN_WARNING "mydriver5: arg %x unsupported in the
SET_MODE ioctl command\n", (int)arg);
            return -EINVAL;
    }
    break;
  case GET_MODE: /* Send my_mode value to user space */
    if (copy_to_user((void*)arg, &my_mode, sizeof(int))) {
      printk(KERN_WARNING "mydriver5: copy_to_user error\n");
      return -EFAULT;
    }
    break;
  default :
    printk(KERN_WARNING "mydriver5: 0x%x unsupported ioctl
command\n", cmd);
    return -EINVAL;
  }
  return 0;
}
```

Il est bien entendu nécessaire d'ajouter la ligne suivante à la définition de la structure `file_operations` utilisée.

```
  .ioctl   =          mydriver5_ioctl,
```

Coté utilisateur, le code qui suit permet de tester la méthode :

```
  for (i = 1 ; i < 4 ; i++) {
    printf ("setting mode= %d\n", i);
    if ((n = ioctl (fd, SET_MODE, i)) < 0)
      perror ("ioctl");
    if ((n = ioctl (fd, GET_MODE, &arg)) < 0)
      perror ("ioctl");
    else
      printf ("arg= %d\n", arg);
  }
```

Ce qui donne à l'exécution :

```
# ./ioctl_test /dev/mydriver5
setting mode= 1
arg= 1
setting mode= 2
arg= 2
setting mode= 3
ioctl: Invalid argument
arg= 2
```

Les files d'attente

Une file d'attente permet à un processus de libérer le processeur lorsqu'il est en attente de données. Le processus pourra être réveillé soit par un signal, soit par l'arrivée des données. L'exemple présenté est dérivé de celui du paragraphe concernant l'allocation dynamique de mémoire.

Nous déclarons tout d'abord une file d'attente.

```
static DECLARE_WAIT_QUEUE_HEAD(read_wait_queue); /* Read wait queue */
```

Dans la méthode `read()`, nous ajoutons la séquence de code suivante, qui indique au processus de s'endormir s'il n'y a pas de données disponibles.

```
  /* Sleep if no data available. */
  ret = wait_event_interruptible(read_wait_queue, num != 0);
  if (ret < 0) {
    printk(KERN_DEBUG "mydriver6: woke up by signal\n");
    return -ERESTARTSYS;
  }
```

Dans la méthode `write()`, nous ajoutons une ligne permettant de réveiller le processus, puisque des données sont alors disponibles.

```
  num = real; /* Destructive write (overwrite previous data if any) */
  /* Wake up one blocked processes in read_wait_queue */
  wake_up_interruptible(&read_wait_queue);
  printk(KERN_DEBUG "mydriver6: wrote %d/%d chars %s\n", real, count, buffer);
```

Nous déroulons la séquence de test. L'écriture puis la lecture des données s'effectuent comme dans l'exemple précédent.

```
# echo -n salut > /dev/mydriver6
# dd bs=5 count=1 < /dev/mydriver6 2> /dev/null
salut
#
```

Par contre, une deuxième lecture des données bloque le `read()`, puisqu'il n'y a plus de données disponibles. On débloque la lecture par un [Ctrl]-[C] ou en effectuant une nouvelle écriture sur le fichier spécial associé depuis un autre terminal.

```
# echo -n 12345 > /dev/mydriver6 2> /dev/null
12345
#
```

Conclusion

Nous avons abordé ici quelques éléments permettant de comprendre la structure des pilotes en mode caractère. Certains points importants, comme la gestion des interruptions, la méthode `mmap()` ou la création de *threads* en espace noyau, n'ont pas été traités. Ils seront abordés lors d'un prochain épisode !

Pierre Ficheux,

pierre.ficheux@openwide.fr

RÉFÉRENCES

- *The Linux kernel module programming guide* sur : **http://www.tldp.org/LDP/lkmpg/2.6/html/index.html**
- RUBINI (Alessandro), *Linux device drivers*, 2ème édition sur : **http://www.xml.com/ldd/chapter/book**
- Un article sur `udev` sur :**http://www.marmottux.org/index.php/2004/10/29/5-udev-ou-comment-remettre-les-peripheriques-en-place**
- « *Driver Porting: device classes* » sur : **http://lwn.net/Articles/31370**
- FICHEUX (Pierre), « API des modules Linux », *Linux Magazine* n° 88.
- FICHEUX (Pierre), « Débogage en espace noyau avec KGDB », *Linux Magazine* n° 88.
- Support de cours Open Wide « Noyau Linux » et exemples par Stelian Pop.
- Code source des exemples présentés dans cet article sur :**http://pficheux.free.fr/articles/lmf/kernel_programming/char_drivers/exemples.tgz**

Le langage Ada – 15 : conteneurs

Yves Bailly

EN DEUX MOTS **Pour Ada comme pour tout langage de programmation, les types fondamentaux deviennent rapidement insuffisants. Des structures plus sophistiquées comme les listes chaînées ou les dictionnaires sont indispensables pour tout développement d'envergure. La norme Ada2005 enrichit la bibliothèque standard de telles structures.**

C'était en effet un manque important du langage Ada95 (c'est-à-dire, le langage Ada tel que défini par la norme de 1995). Chacun réinventait alors dans son coin sa propre roue, ou bien utilisait des ensembles de paquetages tels que Booch [1] ou Simple Components [2]. Situation évidemment loin d'être idéale. Aussi, la dernière version de la norme Ada (2005) inclut-elle désormais un ensemble de structures de données répondant à ces besoins.

Organisation

L'ensemble des conteneurs Ada prend racine dans le paquetage `Ada.Containers`. Celui-ci ne définit rien d'autre qu'une paire de types : `Hash_Type` pour représenter une valeur de hachage (issue d'une fonction de hachage) et `Count_Type` pour représenter le nombre d'éléments dans un conteneur. Ces deux types dépendent de l'implémentation, toutefois `Count_Type` est un type entier modulaire.

On retrouve les grandes familles usuelles des conteneurs, présentées dans le tableau suivant avec leur correspondance en langage C++ (dans le tableau, `AC` représente `Ada.Containers`).

Dans les structures ordonnées, les éléments sont rangés selon une relation de type « inférieur à » (généralement dans un arbre), tandis que les structures non ordonnées reposent sur une fonction de hachage pour placer les éléments dans une table.

Il convient de noter que les classes `unordered_set` et `unordered_map`, déclarées dans l'espace de nommage `std::tr1`, ne sont pas encore normalisées pour le C++.

Voyons maintenant comment tout cela s'utilise.

Le type Vector

Pour commencer, nous allons nous pencher sur le type `Vector`, qui représente un tableau linéaire dynamique. Les fonctionnalités présentées sont généralement également disponibles pour le type `List` (liste doublement chaînée), à l'exception de tout ce qui concerne l'indexation des éléments, la notion d'indice n'existant pas pour une liste.

La déclaration du paquetage est celle-ci :

```
generic
   type Index_Type is range <>;
   type Element_Type is private;
   with function "=" (Left, Right : Element_Type)
      return Boolean is <>;
package Ada.Containers.Vectors is
...
    type Vector is tagged private;
...
end  Ada.Containers.Vectors;
```

Il s'agit donc d'un paquetage générique, ce qui est bien le moins que l'on puisse attendre d'un tel outil. Les paramètres génériques sont :

- `Index_Type` représente le type utilisé comme indice pour ce vecteur ; cela peut être n'importe quel type entier : comme les tableaux prédéfinis, il est possible d'indexer les éléments par des nombres négatifs, le premier indice n'étant pas forcément 1 (ou 0) ;
- `Element_Type` est le type des éléments à stocker ; ce type ne doit pas être limité, ce qui signifie qu'il doit être possible d'affecter des valeurs entre elles par l'affectation `:=` ;

	Ada		C++	
	Type	Paquetage	Classe	En-tête
Liste doublement chaînée	`List`	`AC.Doubly_Linked_Lists`	`std::list`	`<list>`
Vecteur (tableau linéaire)	`Vector`	`AC.Vectors`	`std::vector`	`<vector>`
Ensemble ordonné	`Set`	`AC.Ordered_Sets`	`std::set`	`<set>`
Ensemble non ordonné	`Set`	`AC.Hashed_Sets`	`std::unordered_set`	`<unordered_set>`
Dictionnaire ordonné	`Map`	`AC.Ordered_Maps`	`std::map`	`<map>`
Dictionnaire non ordonné	`Map`	`AC.Hashed_Maps`	`std::unordered_map`	`<unordered_map>`

- Enfin, la fonction "=" est celle qui sera utilisée pour comparer deux éléments, notamment pour la recherche ; la présence de `is <>` à la fin de la déclaration du paramètre signale que celui-ci est optionnel, l'égalité standard étant utilisée par défaut.

Remarquez que le type `Vector` est un type qualifié par `tagged` : on peut donc lui appliquer toutes les techniques de la programmation objet, en particulier le dériver en un nouveau type, l'étendre en lui ajoutant des données ou des opérations.

Voyons immédiatement un exemple simple d'utilisation :

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Containers.Vectors;
procedure Vector_1 is
   package Int_Vects is
      new Ada.Containers.Vectors(Index_Type => Positive,
                                 Element_Type => Positive);
```

Il est tout d'abord nécessaire d'instancier le paquetage générique. Nous créons ici un nouveau paquetage, nommé `Int_Vects`, permettant de déclarer des vecteurs d'entiers positifs indexés par des entiers positifs (pour mémoire, le type `Positive` est défini comme un sous-type de `Integer` limité à l'intervalle `1..Integer'Last`).

```
   vect: Int_Vects.Vector;
begin
   vect.Set_Length(10);
```

Nous pouvons alors déclarer une variable de notre « nouveau » type `Vector`, défini dans notre « nouveau » paquetage `Int_Vects`. L'opération `Set_Length()` définit le nombre d'éléments contenus dans le vecteur, ce nombre pouvant être modifié par la suite.

Le type `Vector` distingue la *taille* de la *capacité*. La taille représente le nombre d'éléments effectivement contenus dans le conteneur. Elle est fixée par `Set_Length()`, donnée par `Length()` et `Is_Empty()` retourne un booléen à `True` ou `False` selon que cette taille est nulle ou non.

Par contre, la capacité représente le nombre d'éléments qui *pourraient* être stockés sans provoquer une nouvelle allocation de mémoire, autrement dit, la taille maximale que l'on peut demander sans que cela donne lieu à une allocation de mémoire supplémentaire (et incidemment, la potentielle duplication des données existantes dans la nouvelle zone mémoire). Elle est fixée par `Reserve_Capacity()` et donnée par `Capacity()`. Si `v` est un vecteur, on a donc toujours `v.Length() <= v.Capacity()`. Notez, par ailleurs, que l'on peut avoir un vecteur vide (`v.Length()` retourne `0`, ou `v.Is_Empty()` retourne `True`) sans que la capacité soit nulle.

```
   for i in 1..10
   loop
      vect.Replace_Element(i, i**2);
   end loop;
```

La boucle précédente remplit le vecteur – ou plutôt, elle remplace les éléments existants par d'autres. En effet, l'appel à `Set_Length()` a implicitement créé 10 éléments non initialisés. Notre tableau est donc déjà plein, sauf que les valeurs nous sont inconnues. Pour placer un élément à un indice `i`, on utilise `Replace_Element()` avec en paramètres l'indice en question et la valeur voulue. Remarquez la précision de la sémantique utilisée : on ne *stocke* pas une valeur dans le vecteur, on *remplace* une valeur existante par une autre.

```
   for i in 1..10
   loop
      Put(Positive'Image(vect.Element(i)));
   end loop;
   New_Line;
end Vector_1;
```

Enfin, on affiche les valeurs contenues dans le vecteur, l'élément situé à un indice donné étant obtenu par l'opération `Element()`.

Opérateurs sur les vecteurs

Le type `Vector` étant simplement privé, il est possible d'affecter entre eux des vecteurs, même de tailles différentes (mais contenant des éléments de même type), par l'opérateur `:=`. Par ailleurs, deux vecteurs peuvent être comparés pour l'égalité au moyen de l'opérateur `=`, ce qui revient à comparer les éléments des vecteurs.

Plus amusant est l'opérateur de concaténation `&`, que nous connaissons déjà sur les tableaux et les chaînes de caractères. Par exemple :

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Containers.Vectors;
procedure Vector_2 is
   package Int_Vects is
      new Ada.Containers.Vectors(Index_Type => Positive,
                                 Element_Type => Positive);
   use type Int_Vects.Vector;
```

Le début est similaire à l'exemple précédent. On a simplement ajouté une clause `use type`, ce qui permet de rendre visibles toutes les opérations associées à un type donné (donc les opérateurs, ce qui nous intéresse ici) sans pour autant rendre visible tout le paquetage dans lequel est déclaré ce type.

```
   procedure Put_Line(v: in Int_Vects.Vector) is
   begin
      for i in 1..v.Length
      loop
         Put(Positive'Image(v.Element(Positive(i))) & " ");
      end loop;
      New_Line;
   end Put_Line;
```

Pour nous simplifier la tâche, on crée une procédure permettant d'afficher le contenu d'un vecteur. Le transtypage (en rouge) est nécessaire, car l'indice de boucle `i` est implicitement de type `Ada.Containers.Count_Type` (le type retourné par l'opération `Length()`), alors que l'opération `Element()` attend un paramètre de type `Positive` (en fait, du type générique formel `Index_Type`).

```
   v1: Int_Vects.Vector;
   v2: Int_Vects.Vector;
begin
   v1 := 1 & 2;
   Put_Line(v1);
```

Premier exemple, la concaténation de deux valeurs produit un vecteur. Naturellement, cela ne fonctionne que si le type des deux opérandes est celui des éléments du vecteur, et que le récepteur du résultat (ici, l'opération d'affectation) est bien du type vecteur ainsi créé.

On pourrait enchaîner ainsi les concaténations, par exemple quelque chose comme :

```
Put_Line(10 & 20 & 30 & 40);
```

mais ce n'est pas recommandé, étant donné que la création puis la destruction des résultats intermédiaires peuvent s'avérer coûteuses.

```
   v2 := v1 & 3;
   Put_Line(v2);
   v1 := 4 & v1;
   Put_Line(v1);
```

Il est également possible d'étendre un vecteur en le concaténant avec une valeur. La première ligne est équivalente à :

```
v2 := v1;
v2.Append(3);
```

tandis que la deuxième (qui modifie `v1` en y plaçant le résultat) est équivalente à :

```
v1.Prepend(4);
```

Les opérations `Append()` (ajouter à la fin) et `Prepend()` (ajouter au début) prennent un paramètre supplémentaire optionnel indiquant le nombre de copies de la valeur ajoutée.

```
   v2 := v2 & v1;
   Put_Line(v2);
end Vector_2;
```

Enfin, il est également possible de concaténer deux vecteurs pour en produire un nouveau, comme le montre le code précédent. Cette commande est équivalente à :

```
v2.Append(v1);
```

Les opérations `Append()` et `Prepend()` peuvent en effet recevoir un vecteur en entrée, plutôt qu'un simple élément.

Ce programme affiche tout simplement :

```
 1  2
 1  2  3
 4  1  2
 1  2  3  4  1  2
```

Il existe également une série d'opérations `Insert()` qui permettent d'insérer une valeur au milieu d'un vecteur, en redimensionnant celui-ci au besoin. Mais notez que ce genre d'action est généralement coûteux, car cela implique un déplacement en mémoire des éléments se trouvant après l'élément inséré. Si vous devez effectuer fréquemment de telles insertions (ou suppressions avec `Delete()`), considérez plutôt le type `List`.

Les curseurs

Nous avons jusqu'ici manipulé les éléments par le biais de leur indice. Mais il existe un autre moyen d'y accéder, disponible sur tous les types de conteneurs : le *curseur*, représenté par le type privé `Cursor`. Il s'agit là de l'équivalent des itérateurs de la bibliothèque standard du langage C++.

Voici un exemple déterminant quelques nombres premiers par l'algorithme du crible d'Ératosthène s'appuyant sur une liste (le programme est prodigieusement inefficace, mais c'est pour l'exemple) :

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Containers.Doubly_Linked_Lists;
procedure Sieve is
   package Int_Lists is
      new Ada.Containers.Doubly_Linked_Lists
         (Element_Type => Positive);
   use type Int_Lists.Cursor;
```

Pour commencer, on réalise l'instanciation du paquetage `Doubly_Linked_Lists`, en ne donnant cette fois que le type des éléments (il n'y a pas d'indice et la fonction de comparaison est celle par défaut). La clause `use type` nous facilitera l'accès au type `Cursor` de ce nouveau paquetage.

```
   procedure Put_Line(l: in Int_Lists.List) is
      c: Int_Lists.Cursor;
   begin
      Put("(");
      c := l.First;
      while c /= Int_Lists.No_Element
       loop
         Put(Positive'Image(Int_Lists.Element(c)));
         c := Int_Lists.Next(c);
      end loop;
      Put_Line(" )");
   end Put_Line;
```

Commençons par une simple procédure affichant la liste. Un curseur est tout d'abord déclaré pour parcourir celle-ci. On lui affecte le résultat de l'opération `First()`, qui donne un curseur « pointant sur » le premier élément de la liste. Puis, on boucle, jusqu'à ce que notre curseur reçoive la valeur `No_Element`.

Cette valeur constante, de type `Cursor`, est déclarée dans tous les paquetages de conteneurs. Elle représente un curseur ne pointant sur rien. Dès qu'un curseur reçoit cette valeur, il ne peut tout simplement plus être utilisé, à moins de lui affecter une nouvelle valeur. On a donc une sémantique assez différente de celle du C++, où on peut avoir un itérateur pointant juste un cran au-delà des bornes d'un conteneur.

L'élément associé au curseur est donné par la fonction `Element()`. Remarquez que comme le type `Cursor` n'est pas qualifié par `tagged`, on ne peut pas utiliser la notation pointée sur une instance de ce type : il est donc nécessaire de préfixer `Element()` par le nom du paquetage concerné.

Le passage à l'élément suivant se fait par la fonction `Next()`, qui retourne un curseur pointant sur l'élément suivant ou `No_Element` si on est déjà à la fin de la liste. Cette fonction existe également sous la forme d'une procédure modifiant son paramètre (passé en mode `in out`).

Voyons maintenant comment nous pouvons manipuler la liste :

```
procedure Prime(l: in out Int_Lists.List;
                p:    out Positive) is
   first: Positive;
   last : Positive;
begin
   first := l.First_Element;
   last := l.Last_Element;
```

Les opérations `First_Element()` et `Last_Element()` retournent respectivement le premier et le dernier élément du conteneur. À ne pas confondre avec `First()` et `Last()`, qui retournent des *curseurs* pointant sur ces éléments.

```
   if first*first > last
   then
      l.Delete_First;
```

`Delete_First()` a simplement pour effet de retirer le premier élément de la liste, qui se trouve ainsi plus courte. L'opération symétrique `Delete_Last()` retire le dernier élément. Ces deux opérations sont également disponibles sur les vecteurs, mais, sur ceux-ci, `Delete_First()` est relativement coûteuse.

```
   else
      declare
         curs: Int_Lists.Cursor;
         next: Int_Lists.Cursor;
         elem: Positive;
      begin
         curs := l.First;
         while curs /= Int_Lists.No_Element
         loop
            next := Int_Lists.Next(curs);
            elem := Int_Lists.Element(curs);
            if (elem mod first) = 0
            then
               l.Delete(curs);
            end if;
            curs := next;
         end loop;
      end;
```

Ce qui précède est le cœur du crible : on recherche tous les multiples du premier élément et on les retire de la liste. On utilise pour cela l'opération `Delete()`, qui prend en paramètre (outre le conteneur concerné) un curseur pointant sur le premier élément à supprimer et éventuellement un nombre d'éléments à supprimer. À l'issu de cet appel, le curseur vaut `No_Element` : dans notre cas, nous devons donc en sauvegarder une copie pour avancer dans la liste.

```
   end if;
   p := first;
end Prime;

lst: Int_Lists.List;
p  : Positive;
begin
   for i in 1..25
   loop
      lst.Append(2*i+1);
   end loop;
   while not lst.Is_Empty
   loop
      Put_Line(lst);
      Prime(lst, p);
      Put_Line(Positive'Image(p));
   end loop;
end Sieve;
```

Le programme se termine en affichant simplement le nombre premier trouvé et l'état de la liste à mesure que le crible avance. On obtient l'affichage suivant :

```
( 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 )
 3
( 5 7 11 13 17 19 23 25 29 31 35 37 41 43 47 49 )
 5
( 7 11 13 17 19 23 29 31 37 41 43 47 49 )
 7
( 11 13 17 19 23 29 31 37 41 43 47 )
 11
( 13 17 19 23 29 31 37 41 43 47 )
 13
( 17 19 23 29 31 37 41 43 47 )
 17
( 19 23 29 31 37 41 43 47 )
 19
( 23 29 31 37 41 43 47 )
 23
( 29 31 37 41 43 47 )
 29
( 31 37 41 43 47 )
 31
( 37 41 43 47 )
 37
( 41 43 47 )
 41
( 43 47 )
 43
( 47 )
 47
```

Ce qui illustre bien les réductions successives de la liste.

Les types ensemblistes

Les types `Hashed_Set` et `Ordered_Set` représentent le concept mathématique d'ensemble, c'est-à-dire une collection de valeurs sans duplication. La différence essentielle est que le premier repose sur une fonction de hachage pour organiser ses éléments, tandis que le second repose sur une relation d'ordre entre les éléments. Par exemple :

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Containers.Hashed_Sets;
with Ada.Containers.Ordered_Sets;
procedure Set_1 is
   function Int_Hash(i: in Integer)
      return Ada.Containers.Hash_Type is
   begin
      return Ada.Containers.Hash_Type(abs i);
   end Int_Hash;

   package Int_Hash_Sets is
      new Ada.Containers.Hashed_Sets
         (Element_Type        => Integer,
          Hash                => Int_Hash,
          Equivalent_Elements => "=");
   use type Int_Hash_Sets.Cursor;
```

Première instanciation, celle du paquetage `Hashed_Sets` pour contenir des entiers. La

fonction de hachage consiste simplement à renvoyer la valeur absolue d'une valeur (paramètre générique `Hash` du paquetage), tandis que l'équivalence entre deux valeurs est obtenue par la simple égalité (paramètre générique `Equivalent_Elements`).

```
procedure Put_Line(hs: in Int_Hash_Sets.Set) is
   curs: Int_Hash_Sets.Cursor;
begin
...
end Put_Line;
```

La procédure précédente affiche le contenu d'un ensemble. Nous ne la détaillerons pas, elle est tout à fait similaire à celle vue pour les listes.

```
package Int_Ordered_Sets is
   new Ada.Containers.Ordered_Sets
      (Element_Type => Integer);
use type Int_Ordered_Sets.Cursor;

procedure Put_Line(hs: in Int_Ordered_Sets.Set) is
   curs: Int_Ordered_Sets.Cursor;
begin
...
end Put_Line;
```

Ensuite on instancie le paquetage `Ordered_Sets`, toujours pour contenir des entiers. Cette fois, il suffit de donner le type des éléments, la fonction de comparaison étant par défaut l'opérateur "<" prédéfini.

Voyons maintenant ce que cela donne, en insérant les mêmes valeurs dans le même ordre dans chacun des deux conteneurs :

```
   hashed : Int_Hash_Sets.Set;
   ordered: Int_Ordered_Sets.Set;
begin
   hashed.Insert(-1);
   hashed.Insert(-2);
   hashed.Insert(1);
   hashed.Insert(2);
   Put_Line(hashed);
   ordered.Insert(-1);
   ordered.Insert(-2);
   ordered.Insert(1);
   ordered.Insert(2);
   Put_Line(ordered);
end Set_1;
```

L'opération `Insert()` permet d'ajouter un élément à un ensemble. Résultat de l'exécution :

```
[ 1 -1  2 -2 ]
[-2 -1  1  2 ]
```

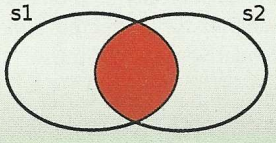
Vous pouvez constater qu'un ensemble issu de `Hashed_Sets` n'est pas ordonné, tandis que `Ordered_Sets` offre des ensembles ordonnés. Savoir lequel des deux types utiliser dépend de vos besoins. D'une manière générale, un ensemble s'appuyant sur une fonction de hachage sera plus efficace pour les opérations d'insertion ou de recherche pour un grand nombre de valeurs, tandis qu'un ensemble basé sur un arbre présente l'intérêt de maintenir l'ordre.

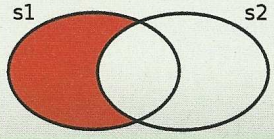
Opérations ensemblistes

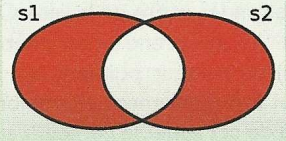
Voyons maintenant quelques opérations communes. Elles sont généralement disponibles sous trois formes : une procédure modifiant son premier paramètre, une fonction de même nom retournant un nouvel ensemble, ou un opérateur surdéfini.

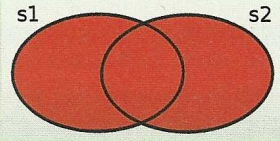
Si `s1` et `s2` sont deux ensembles de même type, les opérations suivantes sont disponibles (données le cas échéant en utilisant la notation traditionnelle et la notation pointée) :

Opération	Sous-programme

intersection
$s1 \cap s2$
 
 	
	Procédure (modifie `s1`) :
	`Intersection(s1, s2);`
	`s1.Intersection(s2);`
	Fonction :
	`s3 := Intersection(s1, s2);`
	`s3 := s1.Intersection(s2);`
	Opérateur :
	`s3 := s1 and s2;`

Différence
$s1 \setminus s2$
 
 	
	Procédure (modifie `s1`) :
	`Difference(s1, s2);`
	`s1.Difference(s2);`
	Fonction :
	`s3 := Difference(s1, s2);`
	`s3 := s1.Difference(s2);`
	Opérateur :
	`s3 := s1 - s2`

Différence symétrique
$(s1 \cup s2) \setminus (s1 \cap s2)$ ou $(s1 \setminus s2) \cup (s2 \setminus s2)$
 
 	
	Procédure (modifie `s1`) :
	`Symmetric_Difference(s1, s2);`
	`s1.Symmetric_Difference(s2);`
	Fonction :
	`s3 := Symmetric_Difference(s1, s2);`
	`s3 := s1.Symmetric_Difference (s2);`
	Opérateur :
	`s3 := s1 xor s2;`

Union
$s1 \cup s2$
 
 	
	Procédure (modifie `s1`) :
	`Union(s1, s2);`
	`s1.Union(s2);`
	Fonction :
	`s3 := Union(s1, s2);`
	`s3 := s1.Union(s2);`
	Opérateur :
	`s3 := s1 or s2;`

Ces opérations ne peuvent s'appliquer qu'entre ensembles de même type et de même nature. Par ailleurs, la norme Ada2005 recommande (mais n'impose pas) que la complexité moyenne des opérations d'insertion, de suppression ou de recherche soit en $O(\log(N))$ pour les dictionnaires de `Hashed_Sets` ou au pire en $O(\log(N)^2)$ pour les dictionnaires de `Ordered_Sets`,

lorsqu'elles impliquent des valeurs. Ces complexités devraient par contre être en O(1) lorsque les opérations sont effectuées par l'intermédiaire de curseurs.

Les dictionnaires

Les dictionnaires permettent de réaliser l'indexation de valeurs d'un type (presque) quelconque par des valeurs d'un autre type (presque) quelconque. Les valeurs indexées sont les *éléments*, les valeurs qui indexent sont les *clefs*. Un exemple typique sont des salaires (valeurs numériques) indexés par des noms (chaînes de caractères). Un dictionnaire est comparable à un tableau, sauf que l'indice n'est pas forcément un nombre entier. Comme les ensembles que nous venons de voir, les types « dictionnaire », nommés `Map`, sont fournis par deux paquetages selon la façon dont sont organisées les clefs :

- `Ada.Containers.Hashed_Maps` pour une organisation utilisant une fonction de hachage sur les valeurs des clefs ;
- `Ada.Containers.Ordered_Maps` pour une organisation fondée sur un ordonnancement des valeurs des clefs.

On pourrait dire qu'un dictionnaire est un ensemble de paires *(clef, valeur)* n'opérant que sur la clef. Par exemple :

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Strings.Unbounded; use Ada.Strings.Unbounded;
with Ada.Strings.Unbounded.Hash;
with Ada.Containers.Hashed_Maps;
with Ada.Containers.Ordered_Maps;
procedure Map_1 is
   package Salaries_Hashed is
      new Ada.Containers.Hashed_Maps(
         Key_Type        => Unbounded_String,
         Element_Type    => Integer,
         Hash            => Ada.Strings.Unbounded.Hash,
         Equivalent_Keys => "=");
   use type Salaries_Hashed.Cursor;
```

On commence par instancier un paquetage `Salaries_Hashed` fournissant un type dictionnaire indexant des entiers `Integer` par des chaînes de caractères `Unbounded_String`. Ce type, déclaré dans `Ada.Strings.Unbounded`, représente une chaîne de caractères dynamique pouvant s'étendre ou se réduire selon les besoins : il est donc beaucoup plus proche du type C++ standard `std::string` que le type usuel `String`, qui n'est en fait qu'un tableau de caractères.

La raison pour laquelle nous utilisons ce type est que dans tous les conteneurs que nous avons rencontrés jusqu'ici, les types stockés doivent être *définis*. Or, le type `String` est déclaré ainsi :

```
type String is array(Positive range <>) of Character;
```

Il s'agit d'un type tableau non contraint : en lui-même, le type `String` ne définit pas les bornes de ce tableau. On dit alors que ce type est *indéfini*. C'est également le cas pour les types « enregistrement » (`record`) présentant un discriminant (revoyez éventuellement le cinquième article de cette série, dans *Linux Magazine* 75).

De tels types indéfinis ne peuvent pas être stockés dans les conteneurs que nous avons vus jusqu'ici (mais nous verrons bientôt comment faire). C'est pourquoi nous utilisons le type `Unbounded_String`, qui est un type défini, le tableau de caractères interne étant manipulé par la mémoire dynamique.

Comme pour le type ensemble déclaré dans `Hashed_Sets`, nous devons fournir une fonction de hachage sur notre type de clefs – c'est-à-dire sur le type `Unbounded_String`. Heureusement, une telle fonction est disponible dans la bibliothèque standard, `Ada.Strings.Unbounded.Hash`. L'équivalence entre clefs est définie par l'égalité entre les chaînes de caractères.

Voyons comment écrire une simple procédure affichant le contenu d'un dictionnaire, tel que défini par notre paquetage :

```
   procedure Put_Line(m: in Salaries_Hashed.Map) is
      crs: Salaries_Hashed.Cursor;
   begin
      Put("{ ");
      crs := m.First;
      while crs /= Salaries_Hashed.No_Element
      loop
         Put("(");
         Put(To_String(Salaries_Hashed.Key(crs)));
         Put(" -> ");
         Put(Integer'Image(Salaries_Hashed.Element(crs)));
         Put(") ");
         Salaries_Hashed.Next(crs);
      end loop;
      Put_Line("}");
   end Put_Line;
```

Comme tous les conteneurs, les dictionnaires possèdent un type `Cursor` permettant de les parcourir. Sauf qu'ici un curseur pointe sur une paire *(clef, valeur)* plutôt que sur une simple valeur. La clef est donnée par la fonction `Key()`, la valeur par la fonction `Element()`. En dehors de cette petite subtilité, la procédure précédente est tout à fait similaire à celles que nous avons déjà vues.

Poursuivons avec un paquetage de dictionnaires ordonnés :

```
   package Salaries_Ordered is
      new Ada.Containers.Ordered_Maps(
         Key_Type     => Unbounded_String,
         Element_Type => Integer);
   use type Salaries_Ordered.Cursor;

   procedure Put_Line(m: in Salaries_Ordered.Map) is
...
   end Put_Line;
```

Il suffit cette fois de donner les types pour les clefs et les valeurs. La procédure d'affichage `Put_Line()` n'est pas détaillée, elle ressemble parfaitement à la précédente.

Utilisons maintenant tout cela :

```
   m_hash: Salaries_Hashed.Map;
   m_ord : Salaries_Ordered.Map;
begin
   m_hash.Insert(To_Unbounded_String("wxyz"), 10);
   m_hash.Insert(To_Unbounded_String("abcd"), 20);
   m_hash.Insert(To_Unbounded_String("mnop"), 30);
   Put_Line(m_hash);
   m_ord.Insert(To_Unbounded_String("wxyz"), 10);
   m_ord.Insert(To_Unbounded_String("abcd"), 20);
   m_ord.Insert(To_Unbounded_String("mnop"), 30);
   Put_Line(m_ord);
end Map_1;
```

L'ajout d'une paire *(clef, valeur)* dans un dictionnaire se fait par l'opération `Insert()`.

La fonction `To_Unbounded_String()` convertit une chaîne de type `String` en une chaîne de type `Unbounded_String`, opération nécessaire étant donné que les écritures littérales comme `"abcd"` sont interprétées comme des chaînes `String`.

Voici enfin ce que ce programme affiche :

```
{ (wxyz -> 10) (abcd -> 20) (mnop -> 30) }
{ (abcd -> 20) (mnop -> 30) (wxyz -> 10) }
```

Comme vous pouvez le constater, on retrouve chez les dictionnaires certaines caractéristiques des ensembles : ceux fondés sur une fonction de hachage stockent leurs éléments dans un ordre quelconque, tandis que ceux fondés sur la comparaison des clefs ordonnent celles-ci.

Sans contraintes

L'exemple précédent a montré que nous ne pouvions stocker dans les conteneurs que des types définis. Cela signifie, par exemple, qu'il est impossible de déclarer une liste de chaînes de caractères de type `String`... du moins avec ce que nous avons vu.

Car c'est en réalité bel et bien possible. Six autres paquetages sont disponibles, dont les noms sont ceux que nous venons de voir préfixés par `Indefinite_`. Ils proposent les mêmes structures de données, mais pouvant contenir des types indéfinis – par exemple, des tableaux non contraints... Voici un exemple d'une liste chaînée de chaînes de caractères, sans passer par le type `Unbounded_String` :

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Containers.Indefinite_Doubly_Linked_Lists;
procedure Indef_List is
   package String_Lists is
      new Ada.Containers.Indefinite_Doubly_Linked_
Lists(
         Element_Type => String);
   use type String_Lists.Cursor;
   procedure Put_Line(l: in String_Lists.List) is
      crs: String_Lists.Cursor;
   begin
      crs := l.First;
      Put("[ ");
      while crs /= String_Lists.No_Element
      loop
         Put("""");
         Put(String_Lists.Element(crs));
         Put(""" ");
         String_Lists.Next(crs);
      end loop;
      Put_Line("]");
   end Put_Line;
   lst: String_Lists.List;
begin
   lst.Append("efgh");
   lst.Append("ijkl");
   lst.Prepend("abcd");
   Put_Line(lst);
end Indef_List;
```

Le paquetage qui nous intéresse ici est `Indefinite_Doubly_Linked_Lists`. Si vous retirez les parties « `Indefinite_` » dans ce qui précède, l'instanciation du paquetage sera refusée par le compilateur, car `Doubly_Linked_Lists` ne peut accepter que des types définis – ce que n'est pas le type `String`.

La procédure d'affichage de la liste n'est pas plus compliquée que précédemment :

Pour enfoncer le clou, voyons un exemple de dictionnaire plus sophistiqué. Supposons que nous devions stocker les patients d'un médecin, indexés par leur nom. Pour chaque patient, on mémorise quelques caractéristiques, maladies, etc. On pourrait commencer ainsi :

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Calendar; use Ada.Calendar;
with Ada.Containers.Indefinite_Hashed_Maps;
with Ada.Strings.Hash;
procedure Indef_Map is
   type Sexe_Type is (Homme, Femme);
   type Patient_Type(Sexe: Sexe_Type) is
   record
      naissance: Time;
      case Sexe is
         when Homme =>
            hydrocele: Boolean;
         when Femme =>
            nb_enfants: Natural;
      end case;
   end record;
```

Le type `Patient_Type` est paramétré selon le sexe du patient (`Sexe_Type`). Selon ce sexe, des informations différentes sont stockées (si vous ignorez le sens de *hydrocèle*, disons qu'il s'agit d'un petit ennui que les femmes ne peuvent pas connaître). Un tel type paramétré est par essence un type indéfini, donc impossible à stocker dans les conteneurs que nous avons vus au début. Mais il existe un type dictionnaire indéfini :

```
   package Patients_Maps is
      new Ada.Containers.Indefinite_Hashed_Maps(
         Key_Type        => String,
         Element_Type    => Patient_Type,
         Hash            => Ada.Strings.Hash,
         Equivalent_Keys => "=");
```

Nous instancions le paquetage `Indefinite_Hashed_Maps`, en donnant comme type de clef le type (indéfini car non contraint) `String` et comme type d'élément le type (indéfini car paramétré) `Patient_Type`. Malgré cette particularité d'avoir clefs et valeurs de types indéfinis, les dictionnaires fournis par ce paquetage s'utilisent tout à fait naturellement :

```
   patients: Patients_Maps.Map;
begin
   patients.Insert("Untel",
                   (Sexe      => Homme,
```

```
                  naissance => Time_Of(1974, 2, 21),
                  hydrocele => False));
   patients.Insert("Unetelle",
                  (Sexe      => Femme,
                   naissance => Time_Of(1980, 4, 3),
                   nb_enfants => 0));
end Indef_Map;
```

Il s'agit là d'un moyen puissant pour construire des structures de données hétérogènes, sans passer par une modélisation objet sophistiquée.

Étant donné les avantages manifestes des conteneurs acceptant des types indéfinis, pourquoi ne pas alors les utiliser constamment ? Et à quoi servent finalement les conteneurs « contraints » ?

La réponse tient à la performance. L'utilisation d'un type défini est plus simple, donc plus rapide, qu'un type indéfini. Prenons un exemple trivial consistant à ajouter des éléments (des entiers) à un vecteur, par l'opération `Append()`, sans réservation préalable. À chaque ajout, le conteneur doit s'agrandir pour accueillir le nouvel élément. Voici le programme tout simple :

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Command_Line; use Ada.Command_Line;
```

Le paquetage `Ada.Command_Line` permet de récupérer les paramètres passés par la ligne de commande.

```
with Ada.Calendar; use Ada.Calendar;
```

Dans `Ada.Calendar` est déclaré un type `Time` représentant une date, ainsi que quelques opérations pratiques pour mesurer un intervalle de temps.

```
with Ada.Containers.Vectors;
with Ada.Containers.Indefinite_Vectors;
procedure Bench is
   package Int_Vectors is
      new Ada.Containers.Vectors(
         Index_Type => Positive,
         Element_Type => Integer);
   package Indef_Int_Vectors is
      new Ada.Containers.Indefinite_Vectors(
         Index_Type => Positive,
         Element_Type => Integer);
```

On instancie les deux paquetages de vecteur, le premier prenant des types définis, le second des types indéfinis. Remarquez que ce dernier peut accepter également des types définis.

```
   nb     : Natural := 10000;
   def    : Int_Vectors.Vector;
   indef  : Indef_Int_Vectors.Vector;
   t_start: Time;
   t_end  : Time;
begin
   if Argument_Count > 0
   then
      nb := Natural'Value(Argument(1));
   end if;
   Put_Line(Natural'Image(nb));
   -- vecteur "défini"
   t_start := Clock;
   for i in 1..nb
   loop
      def.Append(i);
   end loop;
   t_end := Clock;
   def.Clear;
   Put_Line(Duration'Image(t_end - t_start));
```

On affiche d'abord le temps d'exécution (en secondes) de la boucle pour le vecteur de types définis, puis...

```
   -- vecteur "indéfini"
   t_start := Clock;
   for i in 1..nb
   loop
      indef.Append(i);
   end loop;
   t_end := Clock;
   indef.Clear;
   Put_Line(Duration'Image(t_end - t_start));
end Bench;
```

...le temps d'exécution de la boucle pour le vecteur de types indéfinis. Compilé sans aucune optimisation, le remplissage par vingt millions de valeurs nécessite (environ) 1.6s pour le premier cas, 3.6s pour le second : le vecteur « indéfini » est donc deux fois plus lent. Compilé en optimisant par `-O2`, on obtient 0.9s pour le premier cas et 3.1s pour le second : l'écart est donc encore plus important.

Moralité : prenez soin d'utiliser la « bonne » version des conteneurs en fonction de vos besoins.

Conclusion

Voilà pour cette présentation générale des structures de données disponibles dans la bibliothèque standard Ada2005. Comme toujours, de nombreux aspects ont été passés sous silence : consultez le standard du langage pour plus d'informations. Mais vous devriez déjà être capable de construire des structures amusantes, comme un dictionnaire de listes d'ensembles...

La prochaine fois, nous reviendrons sur les types fondamentaux que sont les caractères et les chaînes, que nous utilisons largement sans vraiment connaître leurs possibilités.

Yves Bailly,

http://www.kafka-fr.net

RÉFÉRENCES

- [1] Booch Components : http://booch95.sourceforge.net
- [2] Simple Components : http://www.dmitry-kazakov.de/ada/components.htm

Smalltalk : un modèle pur objet

S. Ducasse, S. Stinckwich

EN DEUX MOTS **Maintenant que nous avons présenté plusieurs aspects de Smalltalk, il est temps de revenir en détail sur le modèle objet de Smalltalk. Ce modèle est simple et ses principes de base sont appliqués de manière uniforme. Cette application uniforme bien que très naturelle est souvent source de confusion chez le novice. Comme tout est objet, en Smalltalk, les classes sont des objets comme les autres. Cet aspect est parfois déroutant.**

Un modèle uniforme et simple

Smalltalk est un langage objet pur. Dans un article précédent, nous l'avions décrit à l'aide des règles suivantes :

- **Règle 1** – Tout est un objet.
- **Règle 2** – Un objet est instance d'une classe.
- **Règle 3** – Une classe définit la structure (variables d'instance) des instances et spécifie les méthodes qui seront exécutées en réponse aux messages envoyés aux instances de la classe. Les variables d'instance sont privées à l'objet lui-même et les méthodes sont publiques.
- **Règle 4** – Une classe peut hériter d'une autre classe par héritage simple.
- **Règle 5** – Les objets communiquent entre eux uniquement par envoi de messages. Lorsqu'un objet reçoit un message, il devient actif : la méthode correspondante est recherchée dans la *classe du receveur*, si elle n'existe pas, la recherche se poursuit dans la super-classe.
- **Règle 6** – Toutes les classes héritent indirectement de la classe `Object`.
- **Règle 7** – Les classes sont les uniques instances de classes appelées méta-classes.

Les règles 1, 2 et 5 du modèle objet de Smalltalk sont illustrées par la figure 1.

Quand un objet reçoit un message, la méthode ayant le sélecteur demandé est recherchée dans la *classe* du *receveur*, puis, si elle n'est pas trouvée, elle est recherchée dans les superclasses. Si on envoie le message

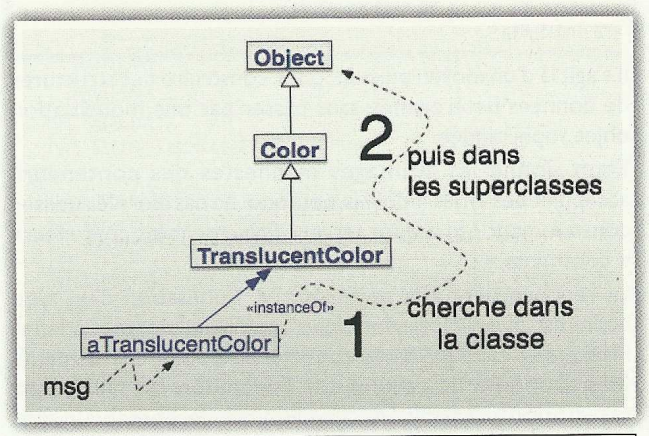


Figure 1 – Recherche d'une méthode dans les super-classes

`brightness` à une instance de `TranslucentColor`, la méthode associée est cherchée dans la classe du receveur donc `TranslucentColor`, puis la classe `Color` où elle se trouve effectivement. Il est intéressant de s'apercevoir que deux liens différents sont suivis : tout d'abord celui d'instanciation, puis celui d'héritage. A priori, rien de bien nouveau ! Mais vous allez voir que les implications bien que naturelles sont parfois déroutantes. Regardons maintenant les conséquences de la règle 7.

Classes et métaclasses

Puisque tout est objet, une classe est aussi un objet. Donc, elle est instance d'une autre classe nommée en Smalltalk sa métaclasse. En Smalltalk une classe est l'unique instance de sa métaclasse. Les métaclasses sont anonymes en Smalltalk : elles portent le nom de leur unique instance, suivi de `class`. La classe `Point` est l'unique instance de la classe `Point class`. La figure suivante illustre ce point : `aTColor` est une instance de la classe `TranslucentColor`. Cette dernière est un objet donc instance de la classe `TranslucentColor class`.

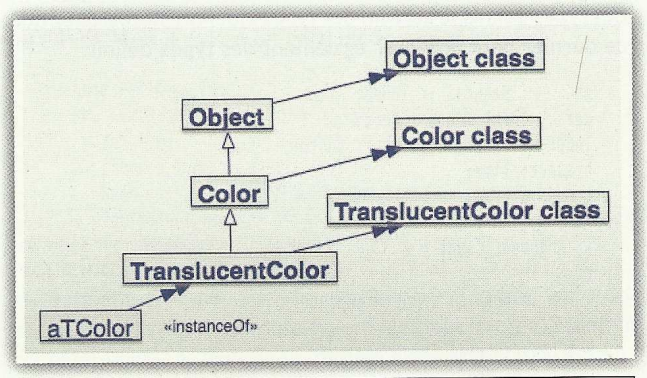


Figure 2 – Relations entre classes et méta-classes

Smalltalk impose une contrainte : la hiérarchie d'héritage des classes et des métaclasses est parallèle comme le montre la figure suivante : `Translucent class` hérite de `Color class` qui hérite d'`Object class`.

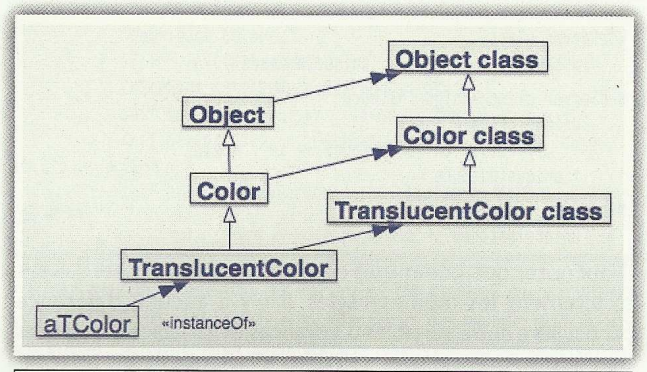


Figure 3 – Les méta-classes héritent les unes des autres

En Smalltalk, il n'existe qu'un seul mécanisme pour invoquer des méthodes et c'est celui que nous avons montré dans la première figure de cet article. Ce mécanisme fonctionne aussi pour les messages envoyés aux classes (car les classes sont des objets comme les autres donc la machine virtuelle ne fait pas de distinction). Lorsqu'un message est envoyé à une classe, ce message est recherché dans la classe du receveur, donc dans la métaclasse. Par exemple, lorsqu'on envoie le message `r:g:b:` à la classe `TranslucentColor` pour créer une couleur, cette méthode est recherchée dans la classe `Translucent class`, puis comme elle n'est pas définie dans cette classe, elle est recherchée dans la classe `Color class`. La méthode à exécuter est toujours recherchée dans la classe du receveur du message, quelle que soit la nature de celui-ci. Il n'y a aucune exception à cette règle.

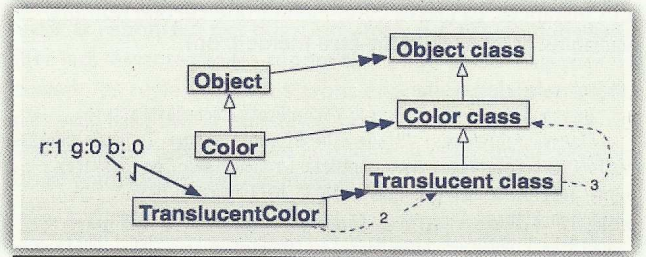


Figure 4 – Recherche d'une méthode de classe

En fait, quand vous modifiez une classe en Smalltalk, vous éditez le code de deux classes distinctes ! Le navigateur de code (*System Browser*) groupe une classe et sa métaclasse. Le bouton « instance/class » permet de passer d'une classe à l'autre. Le bouton « instance » permet d'éditer les méthodes qui s'appliquent sur les instances de classe. Par exemple, `hue` est une méthode d'instance qui est exécutée sur une instance de la classe `Color`. La méthode de classe `wheel:` est une méthode qui est exécutée sur la classe `Color` elle-même (figure 5).

Variables de classes et autres

Il faut ajouter au modèle objet de Smalltalk : les blocs, les variables de Pool, les variables d'instance de classes, les méthodes de classes et les variables de classes. Dans un précédent article, nous avons déjà abordé les blocs qui sont des méthodes anonymes. Nous présentons ici les autres éléments.

Variables de Pool

Les variables de Pool sont des variables qui peuvent être partagées entre plusieurs classes n'étant pas forcément en relation d'héritage. Il est à noter que les variables de Pool et les dictionnaires de Pool qui les contiennent sont un des aspects les moins intéressants de Smalltalk. Néanmoins, nous les présentons ici afin que le lecteur puisse comprendre de quoi il s'agit lorsqu'il parcourt du code.

Alors que les dictionnaires de Pool étaient traditionnellement définis en créant un dictionnaire dans une variable globale, Squeak utilise maintenant des sous-classes de `SharedPool`. Avec la nouvelle manière de définir des `PoolDictionary`, des variables de classes sont utilisées à la place de dictionnaires. Mais encore une fois, nous ne voulons pas entrer trop dans les détails.

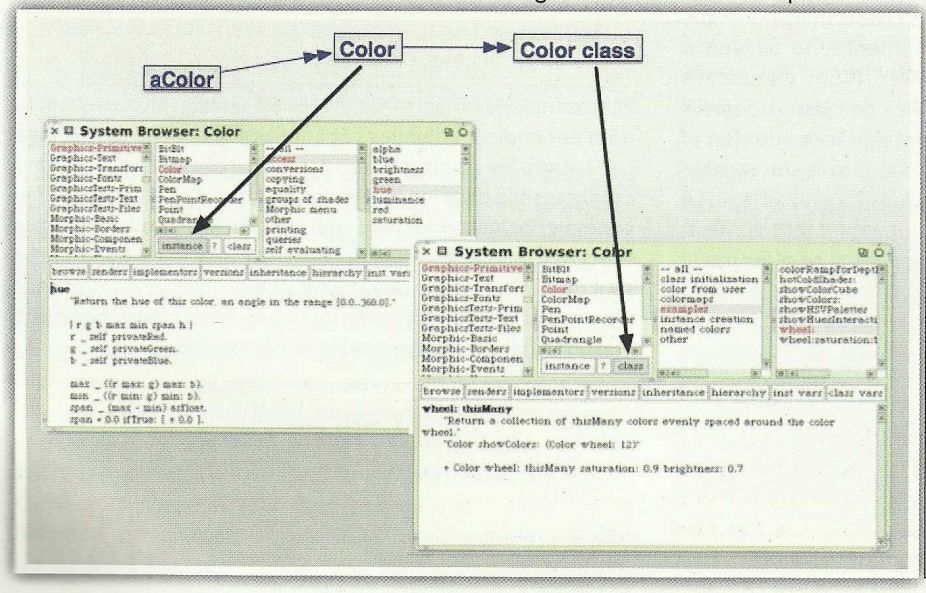


Figure 5 – Vision des deux classes dans le navigateur de classes

Une classe voulant accéder aux variables de Pool doit mentionner, dans sa définition, le dictionnaire de variables de pool qu'elle souhaite utiliser. Par exemple, la classe `Text` indique qu'elle a besoin d'accéder au dictionnaire de Pool nommé `TextConstants` qui regroupe tous les caractères pouvant être utilisés dans des textes. Par exemple, le dictionnaire `TextConstants` possède la clef `CR` qui a pour valeur le caractère `cr`.

```
ArrayedCollection subclass: #Text
   instanceVariableNames: 'string runs'
   classVariableNames: ''
   poolDictionaries: 'TextConstants'
   category: 'Collections-Text'
```

Cela permet aux méthodes de cette classe d'accéder aux constantes directement en référençant les clefs du dictionnaire. Ainsi, on peut écrire la méthode suivante qui rend toujours la valeur vrai (`true`). On voit que l'on peut utiliser directement `CR` dans le code de la méthode.

```
Text>>testCR
      ^ CR == Character cr
```

Il est très rare d'avoir besoin de variables de Pool. Nous vous suggérons de ne pas les utiliser !

Variables d'instances de classes

Les variables d'instance de classes et les méthodes de classes sont simplement des variables d'instances et des méthodes, mais définies sur les classes. Elles sont donc définies en éditant la métaclasse d'une classe, puisqu'elles définissent des attributs et comportements qui seront appliqués sur une classe et non un objet : par exemple, si on souhaite implémenter le schéma de conception (*design pattern*) `Singleton` qui ne permet à une classe de n'avoir qu'une seule instance d'une classe. Imaginons que nous voulions implémenter le Singleton sur la classe `WebServer`. Nous définissons une variable d'instance de classe nommée `uniqueInstance` et des méthodes de classes comme suit, puis nous ajoutons une variable d'instance à la classe `WebServer class`. La méthode de classe `uniqueInstance` vérifie si l'instance a déjà été créée. Si ce n'est pas le cas, une instance est créée et associée à la variable d'instance et, enfin, nous retournons la valeur de cette variable, car dans tous les cas, elle pointe sur la seule instance de la classe.

```
Object subclass: #WebServer
      instanceVariableNames: '  '
        classVariableNames: ''
        poolDictionaries: ''
        category: 'Graphics-Primitives'
```

```
WebServer class
    instanceVariableNames: 'uniqueInstance'

WebServer class>>uniqueInstance
     uniqueInstance isNil
         ifTrue: [ uniqueInstance := self new].
     ^ uniqueInstance

WebServer class>>reset
     uniqueInstance := nil
```

Il faut noter que, comme les classes et les métaclasses suivent exactement les mêmes règles, il n'y a aucune différence particulière entre variables d'instances et variables d'instances de classes. Attention, les variables d'instances sont privées : cela veut dire qu'il n'est pas possible d'accéder aux variables d'instances d'une instance nouvellement créée dans une méthode de classe. De la même manière qu'il n'est pas possible de le faire pour un autre objet. Pour accéder à l'état d'une instance, il faut nécessairement utiliser une méthode.

Variables de classes ou partagées

Le modèle objet de Smalltalk contient une autre sorte de variable nommée en anglais « *class variable* » (variable de classe en français). Pour être plus clair, ces variables auraient dû être nommées variables partagées. Une variable de classe est une variable qui est accessible par toutes les méthodes d'instance et de classe d'une classe et de ses sous-classes. Elle commence par une majuscule pour bien montrer que sa visibilité est plus importante qu'une variable d'instance. Elle permet de définir des valeurs qui ont une durée de vie plus grande que celle des instances. La classe `Color` définit une certain nombre de variables de classes (peut-être même trop).

```
Object subclass: #Color
   instanceVariableNames: 'rgb cachedDepth cachedBitPattern'
   classVariableNames: 'Black Blue BlueShift Brown CachedColormaps
ColorChart ColorNames ComponentMask ComponentMax Cyan DarkGray
Gray GrayToIndexMap Green GreenShift HalfComponentMask
HighLightBitmaps IndexedColors LightBlue LightBrown LightCyan
LightGray LightGreen LightMagenta LightOrange LightRed LightYellow
Magenta MaskingMap Orange PaleBlue PaleBuff PaleGreen PaleMagenta
PaleOrange PalePeach PaleRed PaleTan PaleYellow PureBlue PureCyan
PureGreen PureMagenta PureRed PureYellow RandomStream Red RedShift
TranslucentPatterns Transparent VeryDarkGray VeryLightGray
VeryPaleRed VeryVeryDarkGray VeryVeryLightGray White Yellow'
   poolDictionaries: ''
   category: 'Graphics-Primitives'
```

Par exemple, la variable `ColorNames` est un tableau contenant le nom des couleurs définies par Squeak. Ce tableau est partagé par toutes les instances de `Color` et `TranslucentColor` et il est accessible dans les méthodes de classes et d'instance. Par exemple, la méthode `name` utilise la variable partagée pour retrouver le nom de certaines couleurs. L'idée ici est que comme seul un petit nombre de couleurs peuvent être nommées, il n'est pas nécessaire d'ajouter un champ `name` à chaque couleur, mais de chercher dans la table si nécessaire.

```
Color>>name
     "Return this color's name, or nil if it has no name. Only
returns a name if it exactly matches the named color."
     ColorNames do: [:name | (Color perform: name) = self ifTrue:
[^ name]].
^ nil

Color class>>initializeNames
```

```
        "Name some colors."
        "Color initializeNames"
        ColorNames :=  OrderedCollection new.
        self named: #black put: (Color r: 0 g: 0 b: 0).
        self named: #veryVeryDarkGray put: (Color r: 0.125 g:
0.125 b: 0.125).
        self named: #veryDarkGray put: (Color r: 0.25 g: 0.25
b: 0.25).
        ...
```

Notez que la présence de variables de classes ou variables d'instances de classe pose la question de savoir quand les initialiser. Souvent une initialisation paresseuse est suffisante. Paresseuse signifie que la variable n'est initialisée que la première fois qu'elle est utilisée. Mais cela implique d'utiliser une méthode accesseur et de tester si la variable a été initialisée comme pour la méthode `uniqueInstance`. Une autre façon de procéder est basée sur l'utilisation de la méthode de classe `initialize`. Cette méthode quand elle est définie sur une classe est automatiquement invoquée quand la classe est chargée en mémoire. En la redéfinissant, on garantit ainsi que les variables de classe possèdent bien les valeurs souhaitées.

Démystifions le noyau de Smalltalk

Nous sommes sûrs que certains d'entre vous se sont demandés quelle est la classe d'une métaclasse et que se passe-t-il quand un message de classe est envoyé et qu'il n'est pas défini dans la hiérarchie de classe `TranslucentColor class`, `Color class` et `Object class` ? On peut développer en Smalltalk sans avoir la réponse à ces questions, mais il s'agit de questions fort intéressantes et les réponses n'ont rien de magiques et suivent les règles que nous avons énoncées précédemment. On voit là toute la puissance conceptuelle du modèle de Smalltalk, car il est écrit en lui-même. Il reste donc possible de le comprendre (et éventuellement de le modifier).

Aux règles précédentes, nous ajoutons les règles suivantes :

- **Règle 8** – La hiérarchie d'héritage des métaclasses est parallèle à celle des classes.
- **Règle 9** – Une métaclasse hérite indirectement de la classe `Behavior` via la classe `Class`.
- **Règle 10** – Une métaclasse est instance de la classe `Metaclass`. En particulier, la métaclasse de la classe `Metaclass` est instance de `Metaclass`.

Il faut savoir que la superclasse de la classe `Object class` est la classe `Class`. La classe `Class` regroupe les fonctionnalités liées aux classes comme la création d'instance en Smalltalk. Les méthodes qui sont recherchées dans la superclasse `Object class` vont s'appliquer à des objets qui sont des classes. La superclasse d'`Object class` est la classe `Class`, qui regroupe les fonctionnalités associées à toutes les classes. Par exemple, `TranslucentColor` est une instance de `TranslucentColor class` et si on lui envoie le message : `TranslucentColor new`, `new` sera recherché dans la superclasse de `Object class` donc `Class` et ses superclasses qui regroupent le comportement des classes. Cette méthode sera appliquée à la classe `TranslucentColor`, le receveur du message, et créera une instance comme illustré par la figure suivante.

En Smalltalk, trois classes décomposent plus finement le comportement des classes (création d'objets, définitions de classes, définitions de méthodes...) : `Behavior`, `ClassDescription` et `Class`. Nous obtenons donc le schéma suivant. La superclasse de `Behavior` est `Object`, la racine du graphe d'héritage. `Behavior` est la racine du graphe d'héritage pour les classes.

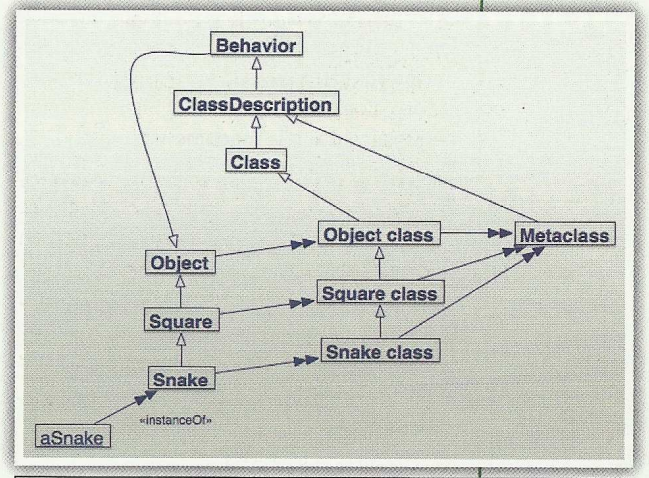


Figure 6 – Où les classes Behavior, ClassDescription et Class eantrent en jeu

`TranslucentColor class` est un objet, donc instance d'une classe. Il nous reste à savoir de quelle classe. `Translucent class` est une classe spéciale, i. e une métaclasse : elle est anonyme et ne possède qu'une instance donc elle est instance de la classe `Metaclass`. C'est ce que la règle 10 énonce. Ainsi, toutes les métaclasses `Color class`, `Object class`, mais aussi `Class class`, `Behavior class` sont des instances de la classe `Metaclass`. La classe `Metaclass` est l'unique instance de la classe `Metaclass class` (règle 7). `Metaclass class` est instance de la classe `Metaclass` (règle 10).

La figure suivante montre l'ensemble du noyau Smalltalk. Non, ne prenez pas peur !

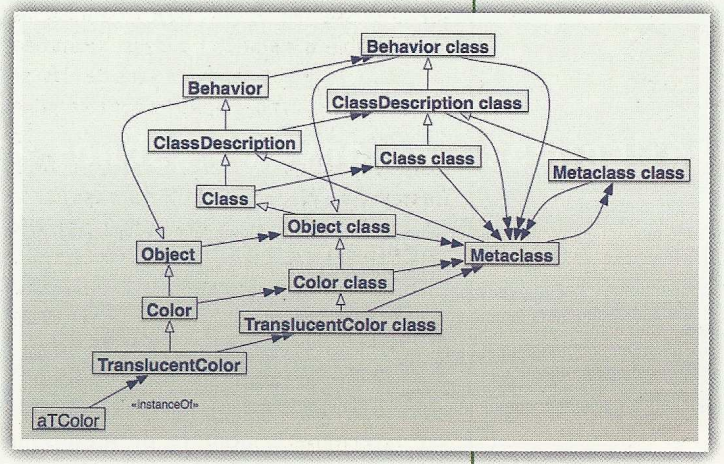


Figure 7 – Où tout se complique !

Ici, on voit l'astuce de la création du noyau que l'on appelle souvent le « *bootstrap* » (amorce en français). Le bootstrap permet à un système de se définir lui-même. Cela peut paraître complexe, mais il n'en est rien : il suffit de suivre les règles que nous avons énoncées ! Les classes `Behavior` et `ClassDescription` représentent le comportement commun entre les classes et les métaclasses. `ClassDescription` introduit les éléments comme les catégories qui permettent au développeur de structurer ses classes, alors que `Behavior` définit le minimum nécessaire pour la machine virtuelle.

Mais à quoi sert tout cela ?

Vous avez pu vous poser cette question au cours de la lecture de cet article. Au-delà de l'explication qui permet de mieux comprendre le modèle objet de Smalltalk et sa simplicité, y-a-t-il de véritables applications aux notions vues ci-dessus ? Oui, par exemple, utilisons ce modèle pour calculer des métriques (c'est-à-dire des mesures) sur le code de Squeak.

Chaque objet Smalltalk peut répondre au message `class` qui lui permet de savoir à quelle classe il appartient :

`1 class.` retourne `SmallInteger`, `1 class class.` retourne `SmallInteger class` et `1 class class class.` retourne `Metaclass`. Enfin, on peut vérifier que : `1 class class class class class == 1 class class class.` retourne `true`. Nous vous laissons chercher pourquoi.

Essayons de répondre maintenant à quelques questions :

- Combien de classes contient une image Squeak ?

`Object allSubclasses size.` retourne 4751. Si on ajoute la classe `Object`, cela nous fait 4752 classes dans l'image que nous sommes en train d'utiliser (ce nombre peut varier en fonction de votre image). Notez qu'ici nous comptons aussi les métaclasses.

- Combien y-a-t-il de métaclasses dans une image Squeak ?

`Metaclass allInstances size.` retourne 2380 métaclasses (instances de `Metaclass`).

- Quelle est la classe qui contient le plus de méthodes ?

```
Object allSubclasses detectMax: [:each |
each methodDict size].
```

retourne `Morph`.

On fait une itération sur toutes les sous-classes de `Object` et, pour chaque classe, on détermine le nombre de méthodes (les méthodes sont conservées dans un dictionnaire dont on calcule la taille). On utilise une itération qui détermine l'élément (la classe) qui a la valeur maximale.

Juste par curiosité, calculons : `Morph methodDict size.` retourne 1165 ! Sans même allez voir le code, on peut conclure qu'une classe qui possède autant de méthodes doit être mal conçue.

En conclusion

Nous avons montré que le modèle de Smalltalk est décrit par un ensemble limité de règles simples. Ainsi, les méthodes et variables d'instances de classes sont des méthodes et variables d'instances définies sur les métaclasses qui ne sont que les classes des classes. Contrairement aux méthodes statiques de Java, les méthodes de classes sont des méthodes normales et suivent les mêmes règles que les méthodes d'instances. On peut utiliser `super` comme dans n'importe quelle méthode. En fait, il n'y a qu'un seul modèle : un objet est instance d'une classe et les méthodes envoyées à cet objet sont recherchées dans la classe de l'objet, et ceci, quel que soit l'objet receveur (même s'il s'agit d'une classe). Nous avons aussi montré comment le noyau Smalltalk suivait des règles très simples. Il est lui-même le résultat d'une modélisation objet !

La classe `Metaclass` spécifie le comportement des métaclasses (instance unique, anonyme). La classe `Class` définit le comportement de toutes les classes. `Metaclass` et `Class` héritent leurs comportements de `ClassDescription` et `Behavior`. `ClassDescription` hérite de la classe `Behavior` qui représente l'information minimale requise par le système pour créer des objets et leur envoyer des messages ainsi que les comportements pour que le programmeur organise les méthodes. Tout ceci peut paraître complexe, mais montre la transparence de Smalltalk par rapport aux concepts mis en avant et leur utilisation par le système lui-même.

Le modèle objet de Smalltalk est principalement utilisé par les outils de développement comme le navigateur de classes pour afficher des informations sur le système et permettre une navigation facile dans les classes et méta-classes.

S. Ducasse et S. Stinckwich,

Stephane.ducasse@univ-savoie.fr
Serge.Stinckwich@info.unicaen.fr

LIENS

- Le site officiel : **http://www.squeak.org/**
- Le *wiki* de la communauté française : **http://community.ofset.org/wiki/Squeak**
- Des livres gratuits en ligne sur Smalltalk et Squeak : **http://www.listic.univ-savoie.fr/~ducasse/FreeBooks.html**
- Un livre sur Squeak en français : **http://www.iam.unibe.ch/~ducasse/Books.html** : BRIFFAULT (X.), DUCASSE (S.), *Squeak*, Eyrolles, 2002.

amen
IN WEB WE TRUST

24 jours, 24 promos*

Vous n'avez pas à nous prier pour que ce soit Noël tous les jours

A partir
du 1er décembre

Chez AMEN, le Père Noël fait des cadeaux exceptionnels aux internautes.

Et tous les jours ça change !

Venez sur le site www.amen.fr et profitez de nos offres uniques sur les noms de domaine, les solutions d'hébergement et les serveurs.

Attention, nos cadeaux changent tous les jours et vous n'avez que 24 heures pour en profiter.

► Avec AMEN, l'Internet devient un paradis et c'est vous qui en profitez !

► Pour plus de renseignements **0 892 55 66 77** (0,34€ / min) ou **www.amen.fr**