GNU LINUX MAGAZINE / FRANCE

France Métro : 6,40€ - DOM 6,95€ - BEL : 7,30€ - LUX : 7,30€ - PORT. CONT. : 7,30€ - CH : 13FS - CAN : 12$ - MAR : 65DH

L 19275 -75- F: 6,40 €

▶SEPTEMBRE ▶2005 ▶NUMERO 75



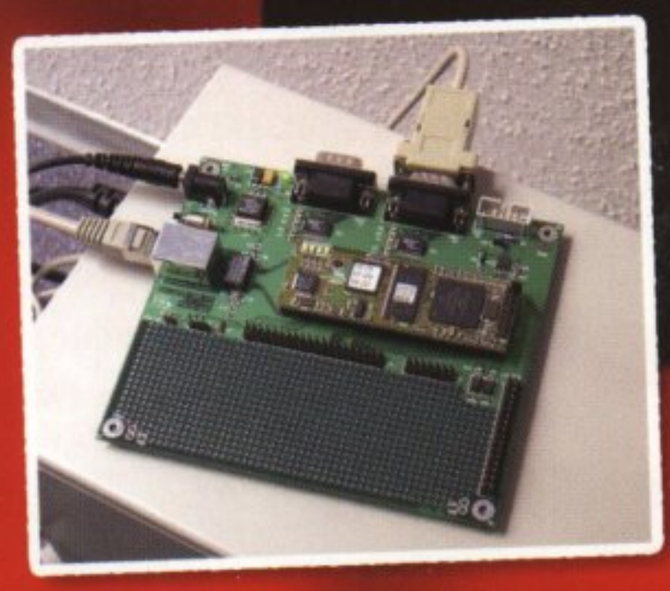


# Tuning de code

**Une balade au coeur du système qui vous fera découvrir le comportement du kernel lors d'accès disque, la manière d'obtenir un code plus performant ou encore l'importance de la mémoire cache et de la taille des tampons**


## Coldfire 5282

**Description et mise en oeuvre d'une carte SSV DIL/Net5280 avec uClinux**



**Administration et développement sur systèmes UNIX**

# → EDITO

Denis BODOR



**EN DEUX MOTS C'est la rentrée...**

**Fini les vacances, la plage, les longues soirées loin de ses serveurs... Là, je ne parle pas tant de moi que de vous, puisque nous n'avons pas chômé durant ces dernières semaines. Comme vous pouvez le constater, la couverture, le rubriquage et la maquette du magazine ont été révisés.**

**L'utilisation des trois colonnes laisse place à une mise en page plus sobre et plus claire. Le principal problème à l'origine du changement était l'aspect et la taille du code. D'autre part, nous avons souhaité « rafraîchir » la maquette graphique et approcher quelque chose de plus épuré.**

**Les rubriques ont également été révisées. Bien que le fond rédactionnel et la motivation des auteurs (et la mienne) ne changent pas, l'organisation par thème ou type de sujet cède la place au bénéfice d'une structuration en fonction de la motivation. Ainsi, le code est omniprésent dans le magazine, mais on fera, par exemple, une différence entre du « Développement » et quelque chose de plus ludique (voir obsessionnel) dans « Hack/Code ».**

**D'autres rubriques que celles utilisées dans le présent numéro existent et feront leur apparition en fonction des articles publiés. C'est le cas de « Science » pour le futur numéro 76 d'octobre dont vous avez un aperçu en page 96. Nous tâcherons désormais de tenir cet « aperçu » au fil des numéros dans la mesure du possible.**

**Je vous laisse à présent découvrir toutes ces petites améliorations et les articles du mois. Vous verrez, les auteurs non plus n'on pas chômés.**

**Rendez-vous dès le 30 septembre en kiosque pour le prochain numéro...**

Denis Bodor


**Linux Magazine France**
est édité par Diamond Editions
B.P. 121 - 67603 Sélestat Cedex
Tél. : 03 88 58 02 08
Fax : 03 88 58 02 09
E-mail : lecteurs@linuxmag-france.org
Service commercial : abo@linuxmag-france.org
Site : www.linuxmag-france.org

Directeur de publication :
Arnaud Metzler

**Rédaction**
Rédacteur en chef : Denis Bodor
**Conception graphique :**
Franck TOUSSAINT
**Responsable publicité :**
Véronique Wilhelm
Tél. : 03 88 58 02 08
**Impression :**
VPM DRUCK / www.vpm-druck.de
**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
Service abonnement :
Tél. : 03 88 58 02 08

PRINTED IN Germany / Imprimé en allemagne /
Dépôt légal :
A parution /N° ISSN : 1291-78 34 / Commission Paritaire : 09 08 K78 976 / Périodicité : Mensuel / Prix de vente : 6,40 €uros


**LÉGENDE :**

- > LIENS
- > REMARQUES
- > ATTENTIONS
- > NOTES
- > ASTUCE
- > ANNEXE

# DEBIAN CORNER

## → Deb'News

Denis Bodor

**EN DEUX MOTS Comme chaque mois, voici les nouvelles de la communauté Debian, des développements et des actions en cours. Nous mettrons également, ici, en avant certains aspects sociaux ou techniques du projet ou tout simplement des points intéressants en rapport avec le projet.**

### Perdition

Lorsqu'on gère un réseau d'utilisateurs, il n'est pas rare de devoir leur offrir un accès à plusieurs serveurs POP3 ou IMAP. En effet, ceux-ci souhaitent souvent relever leurs mails professionnels et personnels.

Plutôt que de mettre en place une solution à base de *fetchmail* centralisant tous les mails ou encore d'autoriser des connexions au niveau du *firewall*, une bonne solution consiste à utiliser `perdition`. Ce paquet permet d'installer un serveur POP3 et IMAP qui redirigera les demandes vers les véritables serveurs. Le choix des serveurs à contacter se fera en fonction de l'utilisateur via regex, LDAP ou un SGBD. `perdition` peut être vu comme un proxy POP3/IMAP sélectif.

On notera au passage que `perdition` peut être utilisé en mode démon mais également via `inetd` et surtout `xinetd`.

*Une entrée existe dans le BTS pour ce paquet à propos d'un problème de violation de protocole Debian.*

### Timeout

Le développement de scripts *shell* est une importante activité de tout administrateur système et de tout utilisateur Unix qui se respecte. Le shell permet de faire un grand nombre de choses, mais malheureusement ne permet pas de faire face aux situations bloquantes.

Ainsi, il n'est pas rare qu'un script reste « coincé » en attente d'une ressource, d'un montage ou d'un hôte. Pour pallier le problème, il suffit d'installer `timeout`. Vous disposerez alors de la commande du même nom prenant en argument un délai en secondes et une commande à lancer. Passé le délai, un `SIGKILL` sera envoyé.

Il est également possible de spécifier un signal à utiliser en lieu et place de `SIGKILL` pour les outils spécifiques. `timeout` s'avèrera extrêmement pratique pour les scripts lancés via `cron`.

*Deux entrées existent dans le BTS pour ce paquet.*

## Une planète Debian pour les francophones

Raphaël Hertzog a annoncé début août la création de **http://planet-fr.debian.net**. L'idée est d'offrir une version francophone de l'équivalant **http://planet.debian.org**. Rappelons que ce site web regroupe les *blogs* des développeurs Debian du monde entier. C'est là qu'on trouve tous les petits papotages et billets d'humeur propres aux développeurs. A présent, la communauté française dispose de son propre serveur dans la langue de Molière et on peut d'ores et déjà y lire bon nombre d'informations intéressantes. Le site comprend, pour l'instant près d'une douzaine d'abonnés mais est ouvert à tous les contributeurs francophones du projet Debian. Il y a donc fort à parier que l'activité et la masse d'informations (croustillantes ?) ira crescendo dans les prochaines semaines.

Pour vous simplifier la vie, sachez que vous pouvez ajouter le fil dans votre agrégateur en utilisant **http://planet-fr.debian.net/rss20.xml**.

## Debian Common Core

DCC (*Debian Common Core*) est le nom d'une alliance destinée à mutualiser les efforts de développement des distributions basées sur Debian. L'initiative lancée par Ian Murdock (Debian et Progeny) a pour but de fournir un noyau de distribution basé sur une Debian stable et les spécifications LSB 3.0. L'avantage est clair. Il s'agit de ne plus perdre de temps et d'énergie dans le développement redondant tel qu'on pouvait l'observer jusqu'à présent. Credativ, KNOPPIX, LinEx, Linspire, MEPIS, Progeny, Sun Wah, UserLinux et Xandros constituent pour l'heure les membres de l'alliance. On remarquera qu'on retrouve l'idée de départ déjà présente dans Debian de construire une distribution minimale pouvant servir de base de travail pour bon nombre de projets. De cette motivation est issue le système de paquets unique à Debian qui fait toute la force de ces distributions dérivées.

Mais les buts de DCC ne sont pas uniquement de l'ordre du développement. On notera qu'il est clairement spécifié dans la FAQ de l'initiative que le projet devra permettre d'accélérer l'adoption de Debian en milieu « commercial ». En effet, la division des efforts n'est généralement pas rassurante pour une entreprise. Ceci s'avère particulièrement vrai pour une distribution sans support officiel ou découlant d'un projet souvent vu comme aléatoire au niveau de la diffusion des versions stables. Ce dernier point est également couvert par les objectifs poursuivis puisqu'ils incluent un fonctionnement en partenariat avec le projet Debian pour s'assurer une plus grande confiance dans le cycle de diffusion. Il est clair que la « prévisibilité » des diffusions de distributions stables est un point important pour tout projet basé sur Debian. Comment faire comprendre à un responsable le cycle de vie et le fonctionnement des mises à jour alors que celui-ci croit Debian obsolète ? Viser un rythme de diffusion prévisible

## Openload

J'ai déjà parlé dans ces colonnes d'outils de test de charge pour serveurs web. En voici un quelque peu différent des autres. Il permet, en effet, de procéder à une mesure en temps réel pour un URL fourni et un nombre arbitraire de clients simulés. Ainsi, directement à l'écran s'affichent diverses informations comme le nombre de transactions par seconde, la vitesse de réponse moyenne durant la seconde écoulée ou encore le pourcentage de code de retour non 200.

La mesure en temps réel permet de travailler à l'optimisation du site, du code PHP ou de la page en question en ayant une vue immédiate sur le résultat. L'optimisation en devient donc très facile pour peu que l'on procède avec un minimum de rigueur.

Pour stopper la mesure, une simple pression sur la touche [ENTREE] suffit. Un résumé et des moyennes pour l'ensemble du test sont affichés. `openload` sera un complément idéal à un outil comme `siege` également disponible en paquet Debian.

*Une entrée existe dans le BTS concernant ce paquet. Il ne s'agit cependant que d'un problème de manpage manquante.*

## asr-manpages

Vous avez sans doute installé les pages de manuel en français et, en bon lecteur de GNU/Linux Magazine, celles concernant le développement (`manpages-dev`). Mais avez-vous pensé aux pages de manuel humoristiques ?

Les paquets `asr-manpages` et `funny-manpages` contiennent respectivement les pages de manuel du groupe `alt.sysadmin.recovery` et d'autres pages toutes aussi amusantes à ne surtout pas prendre au sérieux.

On relèvera des petites perles comme la page sur rtfm « *a response for easy questions from clueless lusers* », le programme guru, slave ou encore axe, « *tools to improve network performance via SNIP* » (SNIP pour « *Sysadmin Network Interrupt Protocol* »).

Rappelons au passage la syntaxe de `man` pour les versions/sections spécifiques : `man 1fun rm`.

Allez, encore une pour la route : xkill, « *extended kill – kill processes or users, including Usenet posters* ».

*Ces paquets disposent d'entrées sans grande importance dans le BTS Debian.*

est quelque chose de très improbable pour tout utilisateur Debian mais ne condamnons pas l'initiative prématurément pour autant. Warren Woodford, CEO et fondateur de la distribution MEPIS, précise clairement les choses : « En établissant des normes et des directives, nous pouvons être le point de contact entre le monde industriel et le monde de Debian et nous pouvons fournir les services que le monde des corporations requiert ».

Certains n'auront pas manqué de relever le grand absent de l'initiative : Ubuntu. La distribution de Canonical est un succès, en particulier auprès des utilisateurs sensibles à l'aspect *desktop*. Souvent jugée comme la distribution la plus viable pour ce domaine, les avis sont cependant partagés sur le côté bénéfique pour Debian. D'autant qu'Ubuntu risque à un moment ou un autre de concurrencer le secteur privilégié de Debian : les serveurs. On notera toutefois que lorsque ce sera le cas, il faudra plus qu'un système d'installation graphique et simplifié à la distribution sud-africaine pour convaincre.

## GNUstep et Debian

Les discussions allaient bon train dernièrement à propos du projet GNUstep et de sa présence dans Debian. En effet, l'environnement GNUstep, implémentation libre d'OpenStep et pseudo-clone d'OS X d'Apple, pose un gros problème dans le respect du FHS (*Filesystem Hierarchy Standard*). Au-delà d'un simple problème de répertoires, c'est un conflit direct entre le FHS et les spécifications d'un projet amont qui se pose. Un environnement OpenStep repose en effet sur un certain nombre de spécifications parmi lesquelles une hiérarchie très spécifique du système de fichiers.

Pour le projet GNUstep, GNU/Linux n'est qu'une plate-forme parmi tant d'autres et les développeurs ne souhaitent donc pas changer quoi que ce soit pour une intégration plus propre vis-à-vis du FHS. De plus, il faut ajouter au crédit des développeurs GNUstep que le projet est bien plus ancien à la notion de standardisation de la hiérarchie du système de fichiers. Il n'en reste pas moins que la situation est bloquante car l'une des rares solutions possibles consistant à placer les éléments dans `/opt` n'est pas compatible avec les règles concernant les paquets officiels Debian bien que conforme au FHS. La question fatidique de la suppression de GNUstep de la distribution n'a pas manqué d'être soulevée tout comme le fait que les paquets GNUstep commencent à être vraiment vieux. Ce n'est pas la première fois que le sujet est mis en avant dans les différentes listes, mais il semble que cette fois le problème prenne des proportions plus importantes. Aura-t-on prochainement la désagréable surprise de voir GNUstep absent de Debian ? C'est bien possible, mais il reste toujours la possibilité au projet GNUstep de produire et maintenir ses propres paquets non officiels.

## apt-key

Si vous êtes utilisateur de la branche *unstable*, vous avez sans doute fait connaissance avec `apt-key`. En effet, la nouvelle version du système Apt (0.6.x) intègre désormais la gestion des signatures GnuPG pour les paquets. Les paquets ayant une signature GnuPG sont considérés comme authentiques (*trusted*). Le système Apt conservera une liste des clefs publiques

et procèdera à une vérification lors de l'installation afin de s'assurer de l'authenticité des données. Lors d'une mise à jour de la liste des paquets, par exemple, une vérification aura lieu et vous devrez posséder la clef publique correspondante. Il est, bien entendu, possible de désactiver cette vérification, mais ce n'est absolument pas recommandable.

Le problème ne se pose pas lors d'une utilisation des paquets et des sources officielles puisque l'ensemble du système sera mis à jour lors du passage de *testing/stable* vers *unstable*. Cependant, si vous utilisez des sources Apt non officielles, vous risquez de rencontrer un problème. C'est le cas, par exemple, des paquets mis à disposition par Christian Marillat. Bien que le fichier `Release.gpg` soit disponible sur le dépôt, `apt-get` ne manquera pas de vous rappeler à l'ordre :

```
W: GPG error: ftp://ftp.nerim.net sarge
  Release: The following signatures couldn't be
  verified because the public key is not available:
  NO_PUBKEY 07DC563D1F41B907
W: You may want to run apt-get update to correct these probleme
```

Vous devrez alors récupérer la clef absente avec :

```
gpg --keyserver pgp.mit.edu --recv-keys 1F41B907
gpg: clé 1F41B907: nom d'utilisateur en double fusionné
gpg: /root/.gnupg/trustdb.gpg: base de confiance créée
gpg: clé 1F41B907: clé publique importée
gpg:       Quantité totale traitée: 1
gpg:                     importée: 1
```

Puis l'intégrer dans la liste des clefs utilisables par le système Apt :

```
gpg --armor --export 1F41B907 | apt-key add -
gpg: no ultimately trusted keys found
OK
```

Reportez-vous aux pages de manuel de `apt-key` et `apt-get` pour en savoir plus.

## dlocate

En tant qu'utilisateur Debian, vous êtes sans doute amené à rechercher une correspondance entre un paquet et certains éléments présents sur le système de fichiers racine. Cette correspondance peut s'établir dans deux sens.

Soit on recherche le paquet dans lequel un fichier donné se trouve (ou un motif), soit on souhaite lister les fichiers installés par un paquet donné. On utilisera normalement `dpkg -S fichier` et `dpkg -L paquet` pour l'une et l'autre opération. `dlocate` vous rendra les mêmes services, mais bien plus efficacement puisqu'il se base sur GNU Locate. L'utilisation d'une base de donnée indexée accélère ainsi grandement les choses. Les opérations avec `dlocate` sont généralement deux fois plus rapides en moyenne qu'avec `dpkg`. `dlocate` offre également un certain nombre d'options intéressantes comme `-du` équivalent à un `du -sck` sur les fichiers du paquet spécifié. La base locate pour les fichiers gérés par le système de paquet est remise à jour quotidiennement via un script *cron* `/etc/cron.daily/dlocate`.

*Ce paquet est référencé bon nombre de fois dans le BTS. Assurez-vous d'y jeter un œil en cas de problème.*

**Denis Bodor,**

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

## ANNONCES DE SÉCURITÉ

**mantis** - missing input sanitising DSA-778
**mozilla** - frame injection spoofing DSA-777
**clamav** - integer overflows, infinite loop DSA-776
**mozilla-firefox** - frame injection spoofing DSA-775
**fetchmail** - buffer overflow DSA-774
**amd64** - several vulnerabilities DSA-773
**apt-cacher** - missing input sanitising DSA-772
**pdns** - several vulnerabilities DSA-771
**gopher** - insecure tmpfile creating DSA-770
**gaim** - memory alignment bug DSA-769
**phpbb2** - missing input validation DSA-768
**ekg** - integer overflows DSA-767
**webcalendar** - authorisation failure DSA-766
**heimdal** - buffer overflow DSA-765
**cacti** - several vulnerabilities DSA-764
**zlib** - remote DoS DSA-763
**affix** - several vulnerabilities DSA-762
**heartbeat** - insecure temporary files DSA-761
**ekg** - several vulnerabilities DSA-760
**phppgadmin** - missing input sanitising DSA-759
**heimdal** - buffer overflow DSA-758
**krb5** - buffer overflow, double-free memory DSA-757
**squirrelmail** - several vulnerabilities DSA-756
**tiff** - buffer overflow DSA-755
**centericq** - insecure temporary file DSA-754
**gedit** - format string DSA-753
**gzip** - several vulnerabilities DSA-752
**squid** - IP spoofing DSA-751
**dhcpcd** - out-of-bound memory access DSA-750
**ettercap** - format string error DSA-749
**ruby1.8** - bad default value DSA-748
**egroupware** - input validation error DSA-747
**phpgroupware** - input validation error DSA-746
**drupal** - input validation errors DSA-745
**fuse** - programming error DSA-744
**ht** - buffer overflows, integer overflows DSA-743
**cvs** - buffer overflow DSA-742
**bzip2** - infinite loop DSA-741
**zlib** - remote DOS DSA-740
**trac** - missing input sanitising DSA-739
**razor** - remote DOS DSA-738
**clamav** - remote DOS DSA-737
**spamassassin** - remote DOS DSA-736
**sudo** - pathname validation race DSA-735
**gaim** - denial of service DSA-734
**crip** - insecure temporary files DSA-733

# → Les Journées Perl 2005

Sébastien Aperghis-Tramoni / Jean Forget

**EN DEUX MOTS Après l'édition précédente qui s'était déroulée à Paris, il était assez naturel d'organiser les Journées Perl 2005 à Marseille. Se déroulant par chance au même moment que l'Austrian Perl Workshop, nous avons pu organiser pour la première fois des transmissions vidéo de présentations entre Vienne et Marseille et, par-là même, permettre à bien d'autres personnes à travers l'Europe de suivre ces deux conférences.**

## Une conférence en province

C'était bien là l'une des difficultés de cette conférence : organisée à la Faculté des Sciences de Luminy, à Marseille, il s'est avéré difficile d'attirer le citadin de la capitale. Plus difficile d'ailleurs que nos amis genevois, qui, bien que n'étant que trois, n'en ont pas moins contribué à la conférence en offrant chacun une présentation. Au total, presque une quarantaine de participants firent le déplacement à Marseille.

L'ensemble des participants a ainsi pu apprécier le cadre agréable de Luminy, le soleil ayant été de la partie. Certains profitèrent d'ailleurs du beau temps pour aller admirer la vue sur les calanques de Sugiton. Paul-Christophe et Stéphane revinrent le samedi pour y descendre et faire trempette.

*La Faculté des Sciences de Luminy*

## Mercredi 8 juin

Sachant que plusieurs « estrangers d'outre-Durance » viendraient la veille pour ne pas arriver trop tard le jeudi matin, une petite bouffe a été organisée dans une pizzeria du Vieux Port. Bien que le rendez-vous avait été donné vers 20h00 - 20h30, certains se sont fait attendre, car ils avaient déjà commencé à écumer les bars, nombreux s'il en est, dans ce coin touristique ;-)

## Jeudi 9 juin

Sachant que certains auraient du mal à venir tôt après une soirée bien arrosée, ce premier jour de conférence ne commence officiellement que vers 10h30, afin de laisser aux participants le temps de s'inscrire et de prendre un petit-déjeuner offert par l'Association Grand Luminy.

Comme pour les précédentes conférences, l'inscription donne droit à une sacoche en tissu à l'effigie du logo des Mongueurs contenant les dépliants du partenaire Grand Luminy et des sponsors Jaguar Network et MailClub.

Les t-shirts sont, cette année, aux couleurs de Marseille, bleu marine pour les organisateurs et blanc pour les participants. La conférence dispose de deux grands amphithéâtres, d'un hall d'entrée et d'une salle de travail, reconvertie en salle de repos où sont disponibles les collations. Les amphithéâtres sont baptisés LMET (Le Monde en Tique) et Jaguar Network, pour remercier deux de nos premiers et plus gros sponsors.

L'entrée offre un véritable petit hall où sont disposées des tables pour réaliser l'enregistrement des participants. Des étudiantes moldaves et bulgares assistent Philippe Blayo, désigné volontaire pour cette tâche. Elles se chargeront ensuite à tour de rôle de veiller sur les collations.

La conférence s'ouvre avec un léger retard par le discours d'accueil du vice-doyen de l'UFR, Jean-Louis Maltret, dans l'amphithéâtre LMET. Après une rapide présentation de la Faculté, il montre ses tout premiers programmes en Perl, sans surprise, des CGI.

Christian Aperghis-Tramoni enchaîne ensuite en racontant des anecdotes de la communauté Perl, agrémentées de photos amusantes de Larry Wall, le gourou en titre.

Vient ensuite Bruno Merlin, d'IntuiLab, qui présente IntuiKit, le toolkit basé sur TkZinc, qui lui-même est un étonnant mélange de Perl/Tk et OpenGL.

Christophe Mertz, lui aussi de la société IntuiLab, avait déjà étonné avec ses démonstrations basées sur TkZinc et InuiKit lors de la conférence *YAPC::Europe 2003* et des *Journées Perl 2004*. Puis Xavier Caron présente ce qu'est le XML, ce qu'on peut faire avec et les différents modules disponibles en Perl pour le manipuler. Au menu, du DOM et du XPath avec `XML::libXML`, du SAX avec `XML::SAX::Machines` et du *twig* avec... `XML::Twig` !

Pendant ce temps, dans l'amphithéâtre Jaguar Network, Stéphane Payrard dispose du reste de la matinée pour présenter Perl 6 et la forte influence des langages fonctionnels sur celui-ci.

Pierre Weis, l'auteur de l'excellent langage fonctionnel OCAML, intervient à plusieurs reprises pour apporter des remarques ou ajouter des précisions.

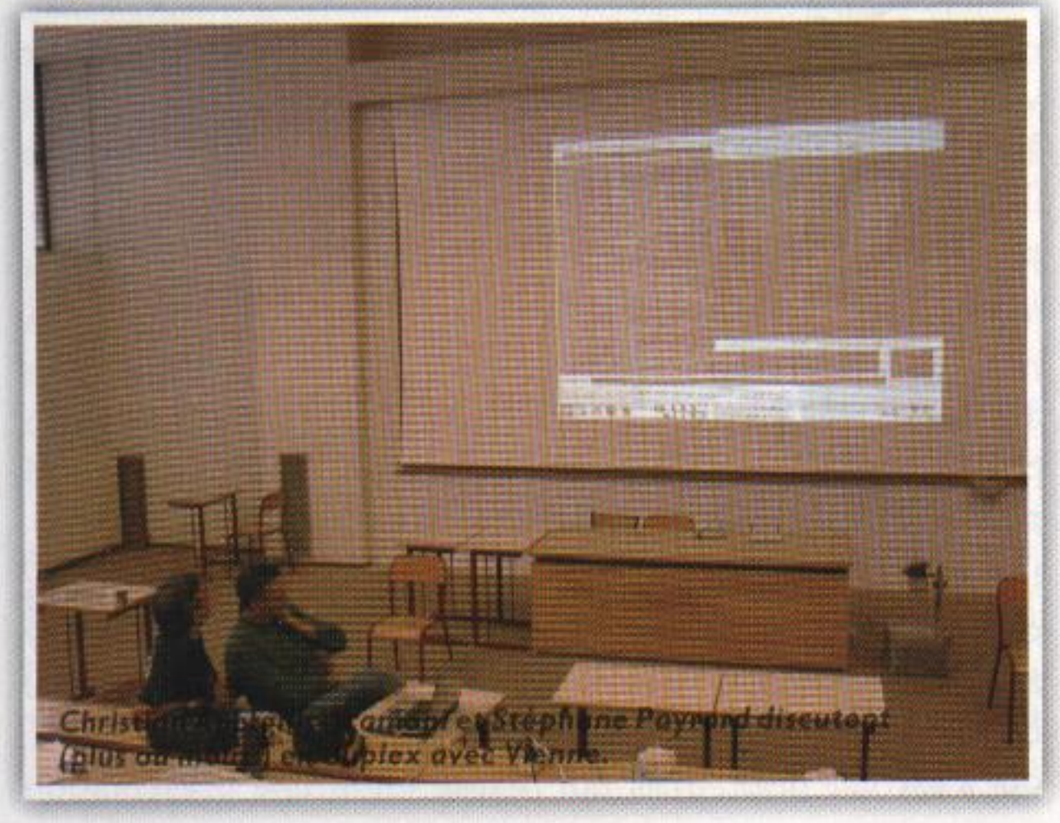
*Autrijus Tang attend de démarrer sa présentation.*

Vient la pause déjeuner. Bien que le campus ne dispose que de restaurants universitaires (pas vraiment connus pour le raffinement de leur cuisine), les participants apprécient de pouvoir grignoter même de simples sandwiches et de lire le mail tout en *chattant* sur IRC au milieu des pins parasols.

La conférence reprend dans l'amphithéâtre LMET avec Scott Lanning qui présente en anglais une démonstration du système de gestion de contenu Bricolage, qui malgré son nom cache un logiciel tout à fait sérieux.

David Morel enchaîne avec une présentation des différentes manières d'interagir avec le système et montre des modules comme `Expect` et `IPC::Run`.

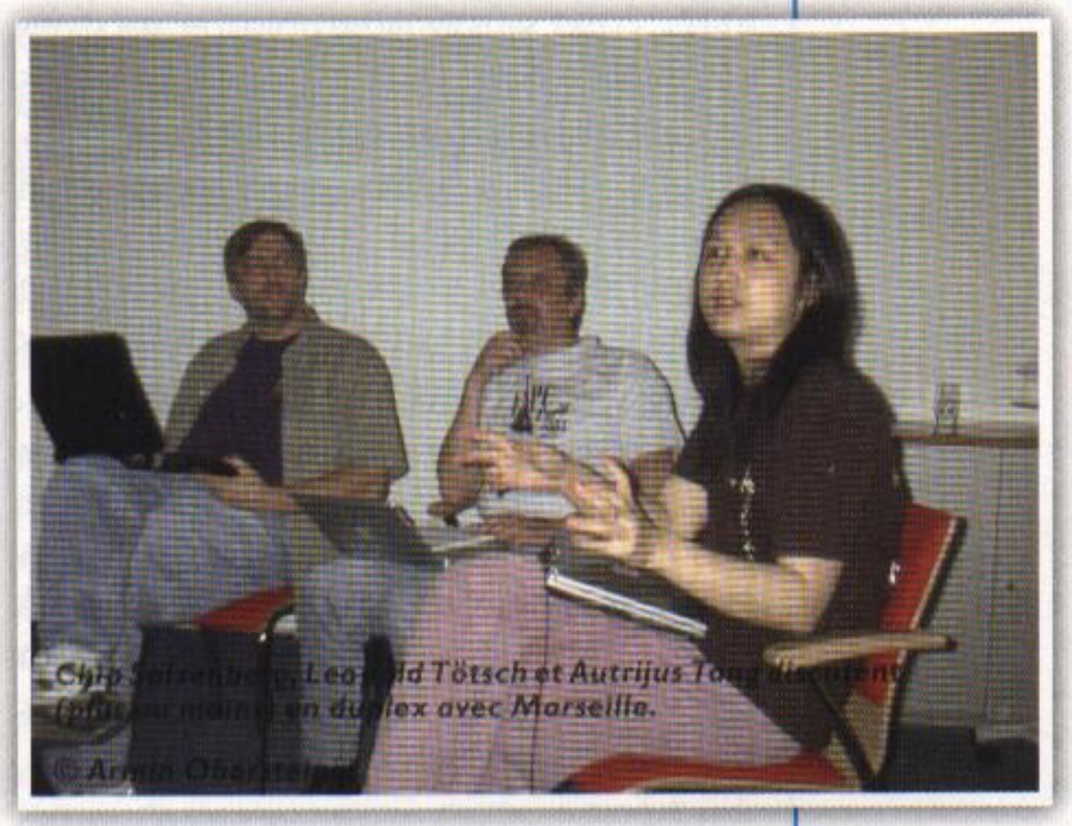
Pendant ce temps, l'amphithéâtre Jaguar Network devient une vraie salle de cinéma avec la retransmission depuis Vienne de la présentation d'Autrijus Tang sur Perl 6 et Pugs. Intitulée « *Apocalypse Now: Perl 6 is here today* », sa présentation est parsemée de références cinématographiques.



*Christian Aperghis-Tramoni et Stéphane Payrard discutent (plus ou moins) en duplex avec Vienne.*

C'est ainsi que nous découvrons que l'auteur de PAR, de Pugs, d'`Acme::ComeFrom` et d'une centaine d'autres modules n'a que 23 ans et qu'il explique très bien et simplement des concepts compliqués à son auditoire.

Après la pause café, Guillaume Rousse nous montre comment il utilise Perl pour analyser des textes et notamment corriger les fautes provenant de la reconnaissance automatique de caractères.



*Chip Salzenberg, Leopold Tötsch et Autrijus Tang discutent (plus ou moins) en duplex avec Marseille.*

© Armin Obersteiner

L'amphi Jaguar reste consacré à la liaison avec Vienne. Se succèdent les présentations de Chip Salzenberg et Leopold Tötsch, où l'architecte en chef et le programmeur principal exposent le développement actuel et futur de Parrot.

S'ensuit un duplex entre Vienne et Marseille où Chip Salzenberg, Leopold Tötsch et Autrijus Tang tentent de discuter avec Christian Aperghis-Tramoni et Stéphane Payrard. « Tentent » seulement car le décalage d'un peu plus de 6 secondes empêche un véritable dialogue, mais permet néanmoins de poser quelques questions... avec l'aide de l'IRC ;-)

La soirée est consacrée au dîner des conférenciers, un repas qui leur est offert par les organisateurs en remerciement de leur participation. Ce repas se tient dans un restaurant de la rue Sainte, juste à côté du Vieux Port. Comme il y a de la place, d'autres personnes peuvent se joindre au dîner, quitte à payer leur part.

C'est ainsi le cas de l'épouse de Cédric Bouvier qui, après une journée de tourisme et de shopping, a néanmoins mangé avec une vingtaine d'informaticiens.

Heureusement, les sujets de discussion n'étaient pas limités à ce seul domaine.

## Vendredi 10 juin

Peu avant le début des conférences, nous avons la surprise de voir apparaître notre sponsor parisien de LMET. Il ne venait pas uniquement pour la conférence, il avait d'autres personnes à voir à Luminy. Mais il profite de son déplacement pour venir nous dire un petit bonjour et pour proposer quelques exemplaires de *Higher Order Perl*.

Ce vendredi commence avec Philippe Bruhat qui présente encore une fois son module `HTTP::Proxy` sans avoir pu préparer ses *slides* et réutilise donc ceux qu'il avait préparés (en anglais) pour sa présentation à *YAPC Europe 2004* à Belfast. Heureusement, il peut se connecter à l'Internet, ce qui lui permet de nous présenter son module en action et montre à titre d'exemple le proxy qu'il utilise pour ses propres besoins.

Suit Cédric Bouvier qui présente un module qu'il a récemment développé, `Distributed::Process`, qui gère le parallélisme et la synchronisation de tâches. Le but recherché est de faire des tests de montée en charge d'un logiciel serveur en lui soumettant des requêtes de processus clients synchronisés.

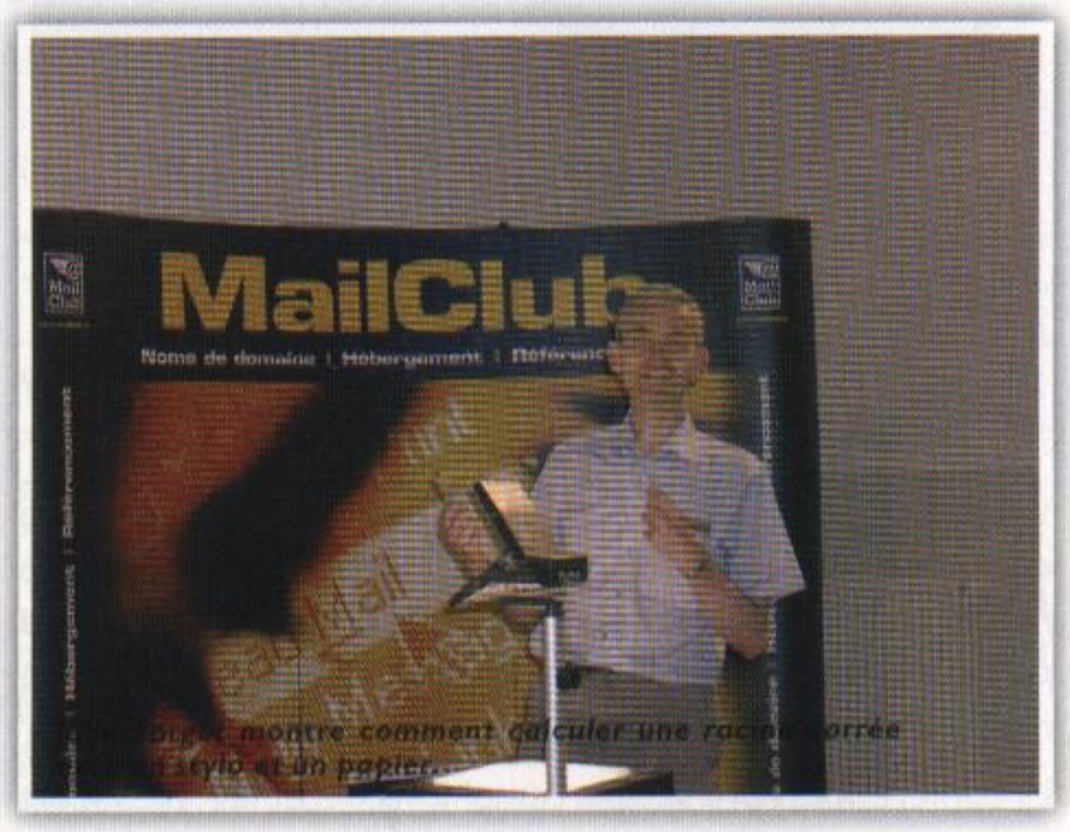
Après la pause, Philippe revient en présentant, sans aucun slide du tout cette fois, l'utilisation de `WWW::Mechanize`. Il essaye de montrer quelques exemples, mais conclut en faisant remarquer qu'il expliquera mieux ceci lors de *YAPC Europe 2005* et prochainement plus en détail dans plusieurs articles de Linux Magazine.

Vient ensuite Laurent Dami qui présente comment effectuer des recherches au sein des fichiers tabulaires à l'aide des modules `File::Tabular` et `Search::QueryParser`.

Après la pause repas, qui s'est transformée pour certains en pause promenade, les conférences reprennent avec de nouveau un duplex avec Vienne.

Cette fois-ci, c'est Christian Aperghis-Tramoni qui présente en anglais l'utilisation de Parrot pour l'enseignement de la programmation en assembleur, exposé qui avait attisé bien des curiosités sur la liste de diffusion **perl6-internals**. Parrot permet en effet aux étudiants d'acquérir les principes de la programmation en assembleur sans pour autant être obligé d'apprendre les arcanes et autres horreurs de l'architecture de la machine cible. Et Parrot tourne sur toutes les architectures répandues et même moins répandues.

Laurent Dami revient avec cette fois une présentation sur plusieurs moteurs d'indexation. Il montre rapidement Plucene, mais indique qu'il ne l'a pas retenu à cause de sa lenteur. Il en vient alors à `Search::Indexer` et présente des exemples d'utilisation. Eric Cholet, qui suit cette présentation par *streaming*, lui fait remarquer par IRC que son module, quoique intéressant, est insuffisant pour traiter les très gros volumes de données auxquels il doit lui-même faire face. C'est enfin au tour de Bruno Merlin de refaire son apparition, mais il constate avec déplaisir que sa première démonstration, bien que visuellement très impressionnante, est trop exigeante pour le vidéoprojecteur. Heureusement, la deuxième démonstration passe correctement et nous pouvons alors découvrir le système qui sera utilisé dans les véhicules de service de l'aéroport de Porto et qui permettra aux techniciens de gérer à la fois leurs déplacements et leur emploi du temps depuis leur véhicule.




## Présentations éclair

Contrairement à l'année passée, les organisateurs ont fait en sorte que cette session de présentations éclair ressemble plus à celle de *YAPC Europe 2003*, en utilisant un décompteur de temps écrit en Perl, un triangle et un grand gong cambodgien. Sébastien Aperghis-Tramoni est le premier à se lancer en présentant les différents sites de recherche du CPAN. Voir la perle de ce numéro pour les détails.

Philippe vient ensuite pour montrer `Acme::MetaSyntactic`, un module parfaitement inutile, donc complètement indispensable, dont le but est de fournir des listes de mots diverses et variées afin de sortir de la banalité des `$foo`, `@bar` et autres `%pipo` (ce module a par ailleurs mis en évidence un bug dans `Devel::Cover`).

Puis vient Jean Forget qui décide de faire une présentation *low tech* sans matériel informatique mais avec un simple rétroprojecteur. Il montre comment extraire une racine carrée avec un papier et un stylo (une règle aurait toutefois été utile pour tirer des traits convenables). Un jour, un module Perl sera capable de simuler les calculs avec papier et stylo.

Laurent Dami montre alors quelques exemples de codes où Perl ne gère pas les références comme on pourrait s'y attendre. Philippe revient pour présenter le bilan annuel du Groupe de Travail « Articles ». Vient alors Pierre Weis, qui s'est fait prier, mais qui réalise enfin une présentation

d'OCAML pour montrer à tous ces jeunots ce que les vrais programmeurs utilisent. Comme il reste des créneaux, Alexandre Buisse improvise une petite présentation où il parle du *garbage collector* de Parrot, qu'il trouve peu efficace et qu'il voudrait remplacer par une version plus moderne, si possible en se faisant sponsoriser par Google (voir les perles pour plus de détails).

Improvisant lui aussi, Stéphane Payrard conclut la session en présentant GreaseMonkey, une extension de Firefox qui permet d'injecter des scripts JavaScript pour modifier l'aspect ou le comportement de pages HTML. Dans certains cas simples, c'est plus pratique que `HTTP::Proxy`.

## Discours de clôture

Sachant que plusieurs participants vont partir prendre leur train, Christian les remercie de leur présence, ainsi que les conférenciers pour leurs présentations. Il remercie aussi les partenaires et sponsors qui ont été nombreux à soutenir la conférence et donne rendez-vous aux participants à *YAPC::Europe 2005*. Philippe, en tant que président de l'association des Mongueurs de Perl, remercie les organisateurs d'avoir réussi l'organisation d'une conférence Perl en dehors de Paris.

## Vente de charité

L'absence de certains habitués aux conférences Perl s'est fait sentir durant la vente de charité, qui a un peu manqué de punch. Ont été mis en vente (avec difficulté) plusieurs livres O'Reilly en français dont *Introduction à Perl*, *Programmation en Perl*, *Programmation avancée en Perl*, *Perl en action*, *Perl DBI*, *Maîtrise des expressions régulières*, *SVG*. Étaient aussi disponibles un *Perl Medic*, *MySQL*, un *Higher Order Perl* vendu par LMET (mais pas aux enchères), deux t-shirts *ActiveState Komodo* et plusieurs t-shirts des précédentes conférences des Mongueurs. Les pièces les plus originales furent un *Perl précis et concis* écrit en bulgare, adjugé pour 55 EUR à Paul Christophe Varoutas et six dédicaces de Mark-Jason Dominus à coller sur son exemplaire de HOP, dont un spécial *use strict 'police'->can('kiss'), my @ass;*.

*Perl 2005. Une partie des participants, conférenciers et organisateurs des Journées*

## Bilan

Malgré une participation moindre que celle espérée et les absences de dernière minute de deux conférenciers (d'où l'annulation de trois présentations), la conférence a été une réussite sur plusieurs points. En premier lieu, les sponsors, qui ont été nombreux à soutenir la conférence, notamment les sponsors et partenaires locaux qui ont beaucoup contribué pour cette édition : Jaguar Network, Evolix, MailClub et Grand Luminy. N'oublions pas les sponsors traditionnels qui, année après année, soutiennent les Mongueurs de Perl : Le Monde en Tique, Linux Magazine, O'Reilly France, ActiveState. Ensuite au niveau des prestations offertes aux participants : dîners, collations et cadre ont été à la hauteur d'une conférence digne de ce nom.

Enfin pour l'innovation majeure de cette édition, à savoir le *streaming vidéo* permettant de relier Vienne et Marseille et autorisant aussi des mongueurs d'un peu partout en Europe (France, Belgique, Pays-Bas, Portugal) à suivre les présentations des *Journées Perl* ou de l'*Austrian Perl Workshop*. On peut remercier pour cela les auteurs du Logiciel libre (français qui plus est) VideoLAN grâce à qui cette tâche est devenue d'une simplicité quasi confondante, y compris sur les systèmes GNU/Linux.

Au final, et malgré les retards et changements de dernière minute du planning des présentations, les *Journées Perl 2005* ont été un beau succès, appréciées tant des participants que des organisateurs.

Merci à tous, et à bientôt pour la conférence européenne *YAPC::Europe 2005* qui se déroulera en septembre à Braga, Portugal, et pour la prochaine édition des *Journées Perl* qui aura probablement lieu à Paris.

### LIENS

Les Journées Perl 2005 :
**http://conferences.mongueurs.net/fpw2005/**

Les supports des présentations :
**http://conferences.mongueurs.net/fpw2005/slides.html**

Austrian Perl Workshop 2005 :
**http://conferences.mongueurs.net/apw2005/**

Galeries de photos d'Armin Obersteiner :
**http://armin.xos.net/bilder/apw_2005/**

YAPC Europe 2005 :
**http://conferences.yapceurope.org/2005/**

**Sébastien Aperghis-Tramoni,**
**Jean Forget,**

**sebastien@aperghis.net,**
**Marseille.pm**

# → Exploration des modules Material et Texture de l'API Python de Blender

Olivier Saraja

**EN DEUX MOTS Ce-mois ci nous continuons l'exploration de l'API Python de Blender, en nous attachant particulièrement aux sous modules Material et Texture et à leur usage au quotidien. Nous verrons donc, principalement, comment créer des matériaux et bien sûr des textures procédurales, puis nous verrons comment associer des indices materau à vos objets.**

Pour bien comprendre la gestion des matériaux via Python, il est nécessaire de bien comprendre celle, plus directe, de Blender même. En effet, Blender permet d'associer à un objet jusqu'à seize matériaux différents, chacun repéré par un indice matériau distinct.

Il convient alors de déclarer pour chaque indice de matériau la liste des faces concernées, mais nous y reviendrons beaucoup plus tardivement dans l'article.

En outre, il est possible de déterminer, en plus de ses caractéristiques propres, jusqu'à dix textures différentes pour un matériau donné. Chaque texture peut alors être associée à un (ou plusieurs !) canal de texture parmi les treize disponibles.

Compliqué ? Pas tant que cela, mais pas forcément facile à mettre en œuvre avec Python. Cet article vise donc à vous donner les bases de la création de matériaux avec l'API Python de Blender. Pour nous y aider, nous effectuerons souvent le parallèle entre une scène créée dans Blender et la même décrite en Python.

## 1. Création d'un nouveau matériau

La définition d'un matériau via l'API Python de Blender est une opération très simple à réaliser, à condition de respecter quelques étapes essentielles. Tout d'abord, il vous sera nécessaire d'importer le module Material dans Python. Comme nous l'avons déjà vu dans les précédents articles, la première ligne de notre code va consister à importer dans Python le module Blender, par la ligne suivante :

```
import Blender
```

la seconde consistant à importer depuis le module Blender les sous-modules souhaités, en l'occurrence Material :

```
from Blender import Material
```

La création d'un nouveau matériau vierge (ou plutôt avec toutes les propriétés par défaut) se fait simplement grâce à la définition d'une commande du type `Material.New()`. Cette variable sera identifiée sous Python sous un nom unique et le nouveau matériau se voit également attribuer un nom unique.

**Syntaxe élémentaire de la création d'un nouveau Materiau :**

`[nom variable matériau] = Material.New('[Nom Blender]')`

`[nom variable matériau]` : il s'agit du nom de la variable Python correspondant au nouveau matériau. Évitez l'usage du point '.' ou des caractères spéciaux au risque de rencontrer des erreurs lors de l'exécution de votre script.

`[nom Blender]` : il s'agit du nom sous lequel le matériau apparaîtra dans Blender

Bien sûr, vous ne souhaitez certainement pas créer via l'API Python toujours le même matériau de base, vous souhaitez pouvoir le définir finement. Nous allons voir ci-après comment

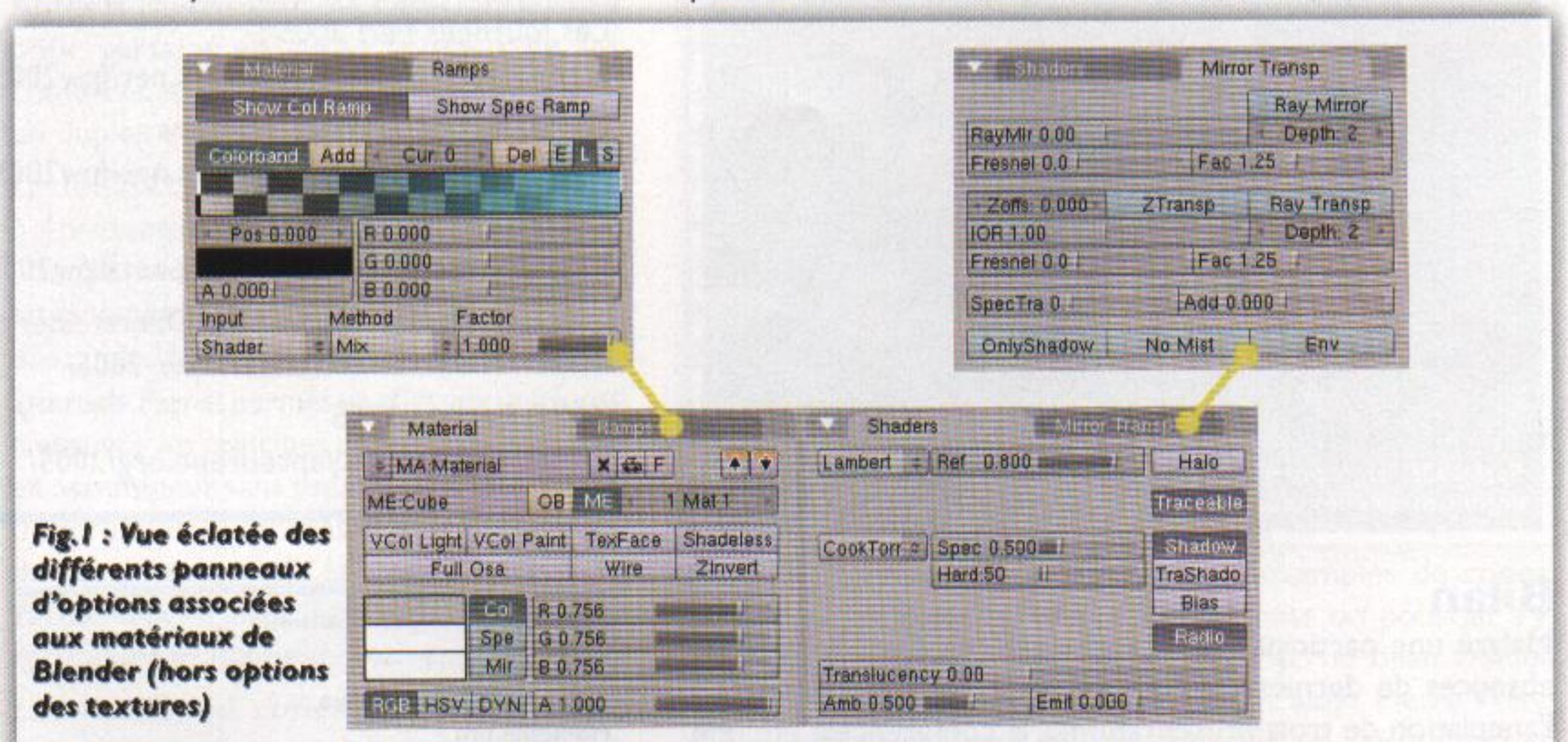



***Fig. 1 : Vue éclatée des différents panneaux d'options associées aux matériaux de Blender (hors options des textures)***

spécifier, en observant les options de Blender, onglet par onglet, la plupart des options les plus intéressantes.

Supposons que nous venons de créer un nouveau matériau, simplement nommé "Material", à l'aide de la ligne suivante :

```
mat = Material.New('Material')
```

Nous allons maintenant explorer les options possibles, mais il sera utile de se rappeler que dans ce qui suit, `mat` est le nom d'une variable librement définie par l'utilisateur et non une fonction au nom imposé par l'API.

**AA REMARQUE**

Dans l'usage des méthodes qui vont suivre, vous avez deux moyens à votre disposition pour déclarer des valeurs. La première est de type `mat.[propriété Blender] = [valeur]`, la seconde de type `mat.set[propriété Blender]([valeur])`. Par exemple :

```
mat.ref = 1.0 et mat.spec = 1.55
```

sont strictement équivalents à :

```
mat.setRef(0.95) et mat.setSpec(1.55)
```

A noter que dans les deux cas, la méthode citée après « `mat.` » commence toujours par une minuscule.

## 1.1 Les options de l'onglet Material

Une petite vue de l'onglet en question n'est pas superflue, la figure 2 est là pour nous aider à visualiser ce qui est possible sous Blender et de voir comment arriver au même résultat avec Python.

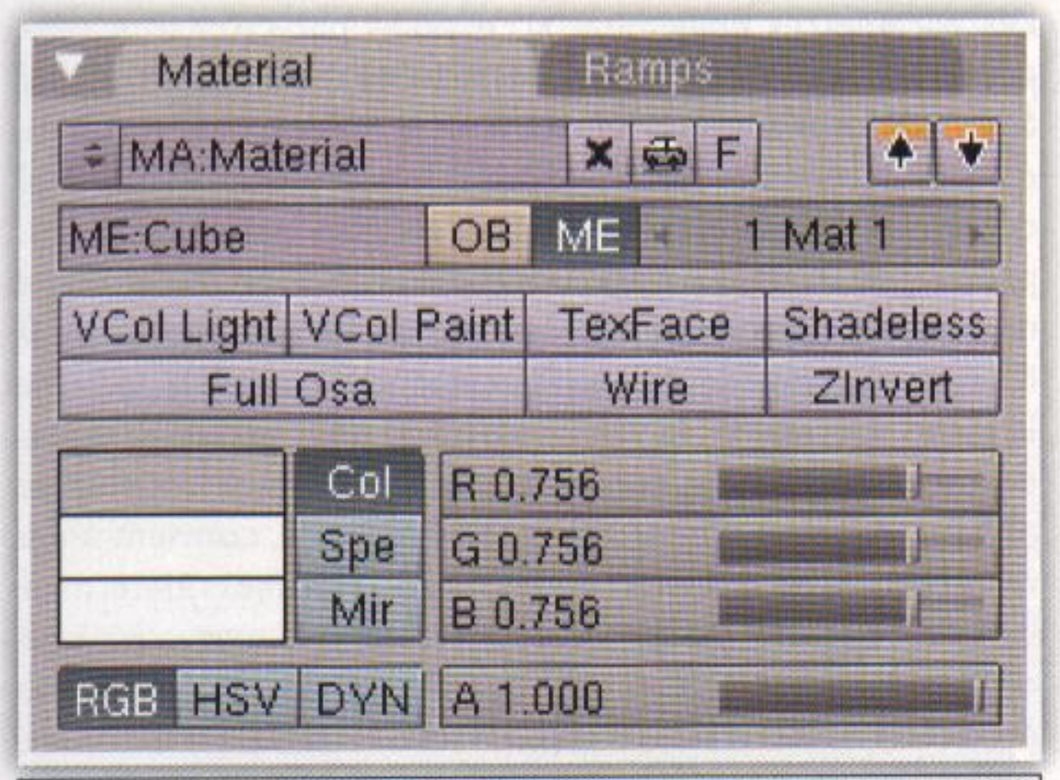



*Fig. 2 : les options de l'onglet Material des boutons Shading (F5) de Blender*

L'une des opérations les plus courantes consiste à définir la couleur de base du matériau. Cela se fait par la définition des trois composantes que sont le rouge (R, Red), le vert (G, Green) et le bleu (B, Blue).

Blender tolère l'usage d'autres méthodes de codage de couleur (HSV et DYN, pour être précis) mais l'API Python ne permet que d'accéder au codage RGB, au moyen de la syntaxe suivante, pour définir les couleurs de base du matériau (Col), de ses reflets spéculaires (Spe) et de ses reflets (Mir), boutons observables sur la figure 2 :

**Syntaxe élémentaire des options du Material :**

`[nom variable matériau].setRGBCol([composante rouge],[composante vert],[composante bleu])` pour la détermination de la couleur de base du matériau (`Col`)

`[nom variable matériau].setSpecCol([composante rouge],[composante vert], [composante bleu])` pour la détermination de la couleur des tâches spéculaires (`Spe`)

`[nom variable matériau].setMirCol([composante rouge],[composante vert],[composante bleu])` pour la détermination de la couleur des reflets (`Mir`)

Il est également possible de définir la composante Alpha (`A`) de l'objet, c'est-à-dire celle qui va permettre de gérer sa transparence.

**Syntaxe élémentaire des options du Material (suite) :**

`[nom variable matériau].setAlpha([valeur])` pour la détermination de la transparence (`A`)

Par exemple, le bloc de lignes suivantes va permettre de créer un matériau de couleur dorée ; malheureusement, il restera de nombreux autres paramètres à définir avant d'obtenir un résultat visuellement attrayant, mais c'est un bon début :

```
mat.setRGBCol([1.0,1.0,0.4)
mat.setSpecCol([1.0,1.0,0.8)
mat.setMirCol([0.8,0.9,0.7)
mat.setAlpha(1.0)
```

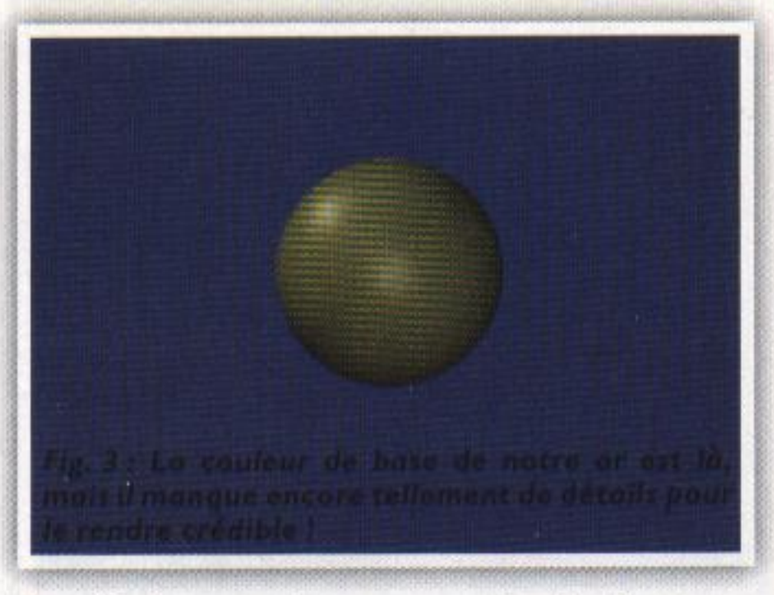


*Fig. 3 : La couleur de base de notre or est là, mais il manque encore tellement de détails pour le rendre crédible !*

Il y a également, dans cet onglet, des boutons poussoirs que nous pourrions avoir besoin d'activer ou de désactiver au rythme de nos expérimentations (en particulier plus tardivement, lorsque nous nous intéresserons au *vertex painting* ou, en français: la peinture sur sommets). Ce sont les boutons *VCol Light*, *VCol Paint*, *TexFace*, *Shadeless*, *Wire*, et *Zinvert*.

Pour les activer, il suffit de spécifier pour le matériau un mode d'affichage, par l'usage très simple de la commande `setMode()` et en spécifiant en guise d'argument le mode qui nous intéresse.

**Syntaxe élémentaire de la définition d'un Mode :**

```
setMode('[mode1]','[mode2]',...)
```

Relativement à cet onglet, il est possible d'employer les arguments suivants pour activer les options correspondantes (d'autres sont disponibles avec le même code, mais nous y reviendrons lorsque nous explorerons les onglets appropriés) : `'Shadeless'`, `'Wire'`, `'VColLight'`, `'VColPaint'`, `'Zinvert'`, `'TexFace'`.

Il est possible de passer à la commande `setMode()` plusieurs arguments, séparés par des virgules. Vous avez d'ailleurs tout intérêt à procéder ainsi, puisque tous les modes non spécifiés seront considérés désactivés. Par conséquent, on peut facilement imaginer que la même commande, sans aucun argument de mode précisé, sert à désactiver par défaut toutes les options. Enfin, ultime remarque, pour préciser un mode, respectez rigoureusement la casse de l'argument (par exemple, `VColLight` et pas `Vcollight`, par exemple) et n'oubliez pas d'encadrer chaque chaîne de caractères par des apostrophes simples : '.

Ainsi, si nous ajoutons la ligne suivante à notre bout de code précédent, nous obtenons une jolie structure en fil de fer doré sur lequel aucune ombre ne se pose :

```
mat.setMode('Wire','Shadeless')
```

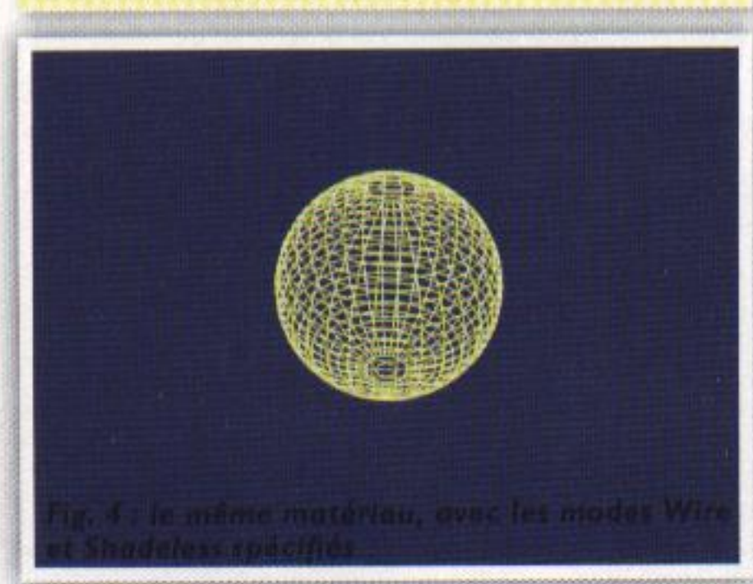


*Fig. 4 : le même matériau, avec les modes Wire et Shadeless spécifiés*

## 1.2 Les options de l'onglet Shaders

Les options découvertes jusqu'à présent nous permettent de définir la couleur de base de notre matériau, mais pas la façon dont il interagit avec l'éclairage ambiant. Pour l'essentiel, c'est l'objectif de cet onglet de Blender : définir les reflets lumineux à la surface des objets pour suggérer une finition (un objet mat ou brillant, avec des reflets durs ou brillants, etc.).

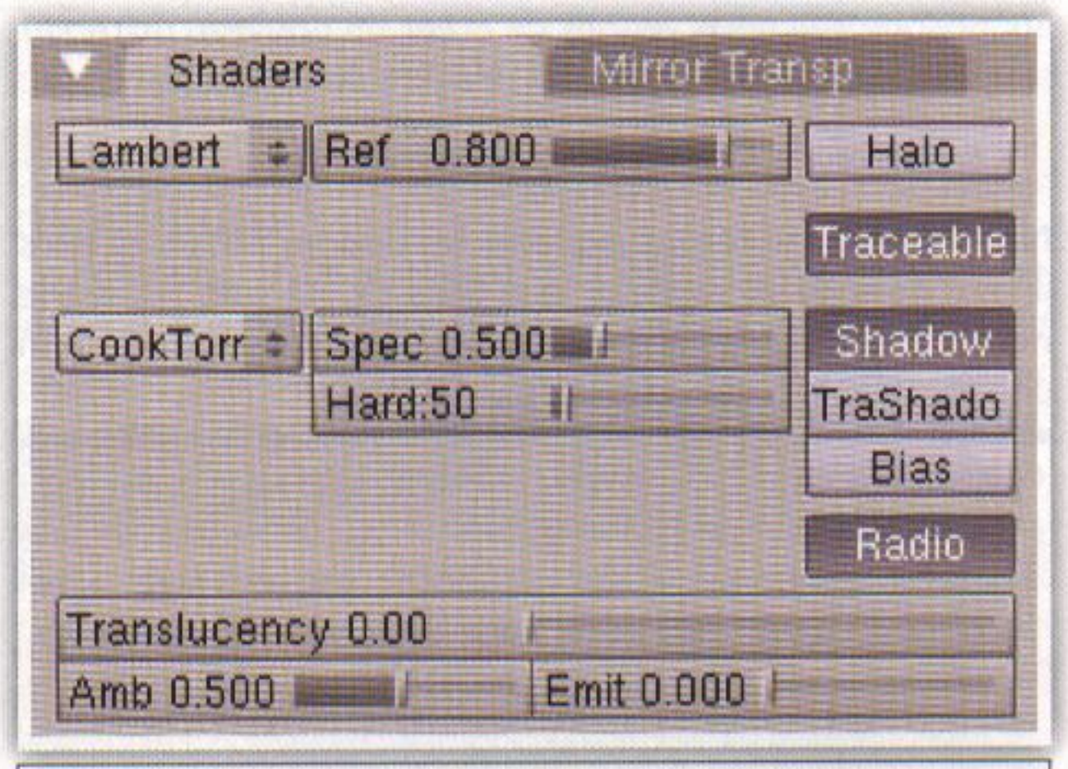



*Fig. 5 : les options de l'onglet Shaders au grand complet*

A noter que Blender propose l'usage de plusieurs algorithmes pour les *shaders* de diffusion (*diffuse shaders* : Minnaert, Toon, Oren-Nayar, Lambert) et de spéculaires (*specular shaders* : WardIso, Toon, Blinn, Phong, CookTorr) mais malheureusement seuls ceux par défaut sont manipulables au travers de l'API Python : Lambert pour le diffuse shader et CookTorr pour le specular shader. Il est dommage que l'accès aux différents algorithmes soit ainsi (pour l'instant, l'API étant en constante évolution) limité mais cela simplifie considérablement la description des méthodes de l'API relatives à cet onglet, chaque shader pouvant avoir ses propres paramètres supplémentaires.

Les paramètres finalement accessibles par l'API sont ceux présentés sur la Figure 5 ci-dessus. Les trois premiers (*Ref*, *Spec* et *Hard*) suggèrent justement la finition d'un matériau, dans la façon dont celui-ci capte la lumière : *Ref* (*Reflection*) détermine la quantité de lumière reflétée par le matériau ; proche de 1, et les conditions d'illumination de la scène mises à part, l'objet sera très clair, tandis que proche de 0, il sera très sombre. *Spec* (*Specular*) détermine la taille de la tâche spéculaire ; la valeur associée peut varier de 0 (pas de tâche spéculaire, convient à des matériaux totalement mats) à 2 (tâche d'intensité très supérieure, convient à des matériaux très brillants). Enfin *Hard* (*Hardness*)détermine la « dureté » de la tâche spéculaire ; sa valeur peut varier de 0 (bords de la tâche extrêmement flous, convient à des matériaux poreux ou à la surface sale ou irrégulière, comme le caoutchouc) à 255 (bords extrêmement durs et nettement dessinés, convient à des surfaces très lisses, vernies ou très dures, comme le verre).

**Syntaxe élémentaire des options des Shaders :**

`[nom variable matériau].setRef([valeurRef])` pour la détermination de la réflection (`Ref`)

`[nom variable matériau].setSpec([valeurSpec])` pour la détermination du spéculaire (`Spec`)

`[nom variable matériau].setHard([valeurHard])` pour la détermination de la dureté du spéculaire (`Hard`)

Il y a trois autres paramètres dans cet onglet que nous pourrions souhaiter donner aux matériaux créés à l'aide de Python. Il s'agit de la translucidité (`Translucency`), de la susceptibilité à la couleur ambiante (`Amb`) et de l'émittance (`Emit`). Le premier détermine la proportion de luminosité qui traverse la paroi externe du matériau, un peu comme s'il s'agissait d'une fine membrane, le tissu d'un abat-jour ou tout simplement celui d'une robe (jamais entendu parlé du soleil vénitien ?). Le second permet de mêler à la couleur propre de l'objet une couleur définie comme étant celle de l'environnement (il s'agit d'une ancienne technique pour simuler, de façon très primitive il faut l'avouer, l'illumination globale). Le dernier permet de déterminer avec quelle puissance le matériau a la particularité de s'auto-illuminer même en l'absence d'éclairage (particulièrement utile dans la détermination d'une solution de radiosité). Malheureusement, de ces trois propriétés, seules deux sont couvertes par l'API Python, la translucidité n'étant à ce jour pas accessible par ce moyen.

**Syntaxe élémentaire des options des Shaders (suite) :**

`[nom variable matériau].setAmb([valeurRef])` pour la détermination du facteur ambiant (`Amb`)

`[nom variable matériau].setEmit([valeurSpec])` pour la détermination de l'émittance (`Emit`)

Ces deux méthodes admettent des paramètres variant de 0.0 à 1.0.

Il y a également des boutons poussoirs qu'il est possible d'activer : *Halo*, *Traceable*, *Shadow*, *TraShado*, *Bias*, *Radio* grâce à la méthode `setMode()` comme nous l'avons vu lors de l'exploration de l'onglet *Material*, en spécifiant simplement, en guise d'argument, le mode qui nous intéresse: `'Traceable'`, `'Shadow'`, `'Halo'`, `'Radio'`. Il n'y a toutefois pas d'argument permettant d'activer les boutons *TraShado* et *Bias*, l'API Python étant toujours en cours de développement.

Pour reprendre notre exemple de tout à l'heure, le bloc de lignes suivant va nous permettre de donner à notre matériau de couleur dorée une finition d'aspect plus métallique :

```
mat.setRef(0.8)
mat.setSpec(1.4)
mat.setHard(20)
```

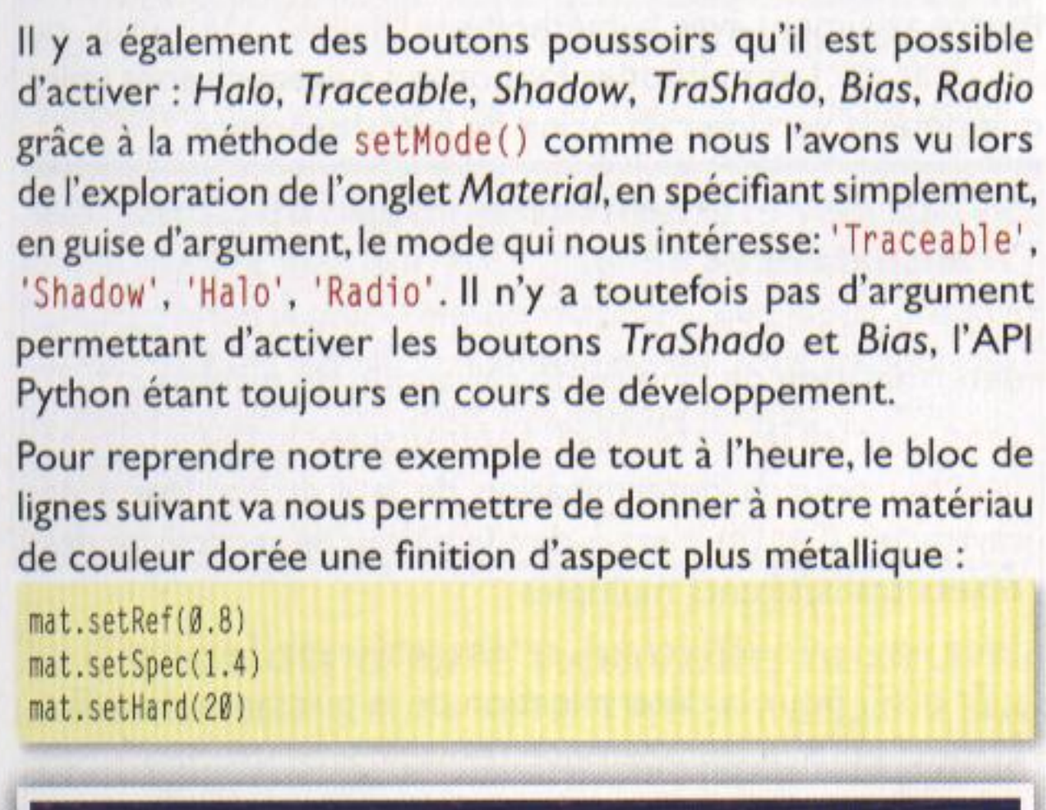


*Fig. 6 : Les tâches spéculaires sont là, ainsi que seul un éclairage suffisamment riche peut le révéler.*

A noter toutefois que dans Blender, lorsque vous activez le bouton *Halo*, vous accédez à de nouveaux paramètres. Ceux-ci sont également accessibles au travers de l'API Python. L'activation du bouton poussoir *Flare* entraîne lui-même l'apparition de nouveaux paramètres, de sorte que la configuration la plus complexe est celle de la figure 7.

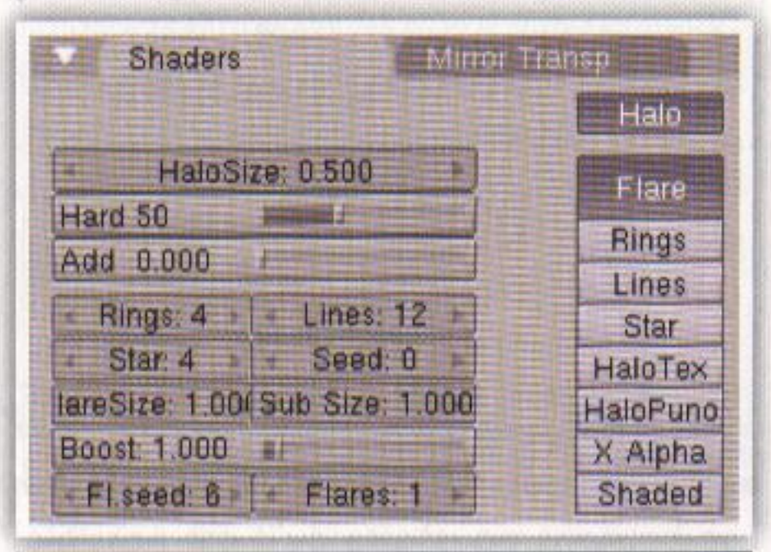



*Fig. 7 : les options relatives aux halos*

Les méthodes accessibles via l'API Python sont résumées dans l'encadré suivant. Par manque de place dans cet article, nous vous renvoyons à la documentation de Blender pour connaître leurs effets.

**Syntaxe élémentaire des options des Halos :**

`[nom variable matériau].setHaloSize([valeurHaloSize)` pour la détermination de la taille du Halo (`HaloSize`)

`[nom variable matériau].setAdd([valeurAdd])` pour la détermination du facteur d'éblouissement (`Add`)

`[nom variable matériau].setNRings([valeurNRings])` pour la détermination du nombre d'anneaux (`Rings`) (nécessite l'activation du mode `'Rings'`)

`[nom variable matériau].setNLines([valeurNLines])` pour la détermination du nombre de lignes (`Lines`) (nécessite l'activation du mode `'Lines'`)

`[nom variable matériau].setNStars([valeurNStars])` pour la détermination du nombre d'étoiles (`Star`) (nécessite l'activation du mode `'Star'`)

`[nom variable matériau].setHaloSeed([valeurHaloSeed])` pour la détermination de la graine pseudo-aléatoire des halos (`Seed`)

`[nom variable matériau].setFlareSize([valeurFlareSize])` pour la détermination de la taille de l'effet lumineux (`FlareSize`) (nécessite l'activation du mode `'Flare'`)

`[nom variable matériau].setSubSize([valeurSubSize])` pour la détermination de la taille des points, cercles et effets lumineux secondaires (`Sub Size`) (nécessite l'activation du mode `'Flare'`).

**Syntaxe élémentaire des options des Halos (suite) :**

`[nom variable matériau].setFlareBoost([valeurFlareBoost])` pour la détermination de la puissance de l'effet lumineux (`Boost`) (nécessite l'activation du mode `'Flare'`)

`[nom variable matériau].setFlareSeed([valeurFlareSeed])` pour la détermination de la graine pseudo-aléatoire des effets lumineux (`Fl.seed`) (nécessite l'activation du mode `'Flare'`)

`[nom variable matériau].setNFlares([valeurNFlares)` pour la détermination du nombre d'effets lumineux (`Flares`) (nécessite l'activation du mode `'Flare'`)

Les boutons poussoirs qu'il est possible d'activer sont les suivants : *Halo*, *Flare*, *Rings*, *Lines*, *Star*, *HaloTex*, *HaloPuno*, *X Alpha*, *Shaded*. Comme précédemment, référez-vous à la documentation de Blender pour une description complète. Il est possible de les activer grâce à la méthode `setMode()` comme nous l'avons déjà vu à deux reprises, en spécifiant cette fois, en guise d'arguments, les modes qui nous intéressent parmi : `'Halo'`, `'HaloFlare'`, `'HaloRings'`, `'HaloLines'`, `'HaloStar'`, `'HaloTex'`, `'HaloPuno'`, `'HaloXAlpha'`, `'HaloShaded'`.

## 1.3 Les options de l'onglet Mirror Transp

C'est ici que se cachent les principales options ayant trait au *raytracing* dans Blender : la gestion des effets de reflet et de transparence. A noter toutefois que la transparence peut être gérée à la fois par le *raytracer* et par le *scanliner* tous deux intégrés à Blender.

Il en résulte la possibilité d'utiliser soit le *Ztransp* (scanliner) soit le *Ray Transp* (raytracer). Dans le premier cas, les paramètres optionnels (*IOR*, *Depth*, *Fresnel* et *Fac*) sont totalement inutiles car non pris en compte.

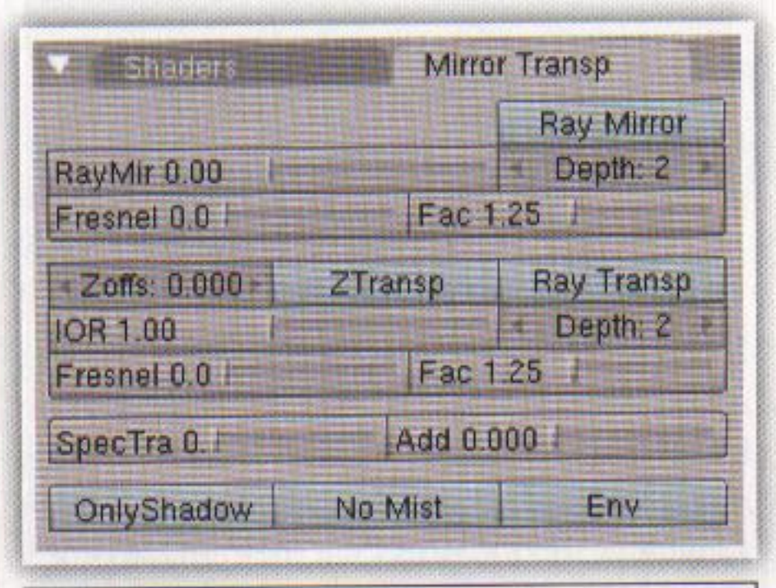



***Fig. 8 : les options relatives au raytracing***

*Raytracing : Technique de rendu d'image consistant à retrouver le trajet naturel des rayons lumineux depuis la source lumineuse jusqu'à la caméra.*

Nous allons commencer par étudier la mise en place de reflets. Cela passe dans un premier temps par la déclaration d'un mode supplémentaire, figuré dans l'onglet *Mirror Transp* par un bouton *Ray Mirror*. Pour la commande `setmode()`, il est nécessaire de lui passer, comme argument, le paramètre `'RayMirr'`. Nous avons ensuite quatre méthodes Python avec lesquelles jouer, ainsi que le précise l'encadré suivant.

**Syntaxe élémentaire des options de Reflets :**

`[nom variable matériau].setRayMirr([valeurRayMirr)` pour la détermination du taux de réflection (`RayMir`).

`[nom variable matériau].setRayMirrDepth([valeurRayMirrDepth])` pour la détermination de la « profondeur » de raytracing (`Depth`), c'est-à-dire le niveau de récursivité des reflets multiples.

`[nom variable matériau].setFresnelMirr([valeurFresnelMirr])` pour la détermination de la puissance de l'effet Fresnel pour les reflets (`Fresnel`).

`[nom variable matériau].setFresnelMirrFac([valeurFresnelMirrFac])` pour la détermination du facteur de Fresnel (`Fac`).

La mise en place de la transparence ne présente guère plus de difficulté, à part que vous avez le choix entre activer la transparence en mode scanline (bouton poussoir *ZTransp*) ou en mode raytracing (bouton poussoir *Ray Transp*). Le choix entre les deux versions se fait par l'usage de l'un ou l'autre argument avec la méthode `setMode()` : `'ZTransp'` ou `'RayTransp'`. Les méthodes Python qui suivent ne sont utiles que pour la version raytracing de la transparence.

**Syntaxe élémentaire des options de Transparence :**

`[nom variable matériau].setIOR([valeurIOR)` pour la détermination de l'indice de réfraction du matériau (`IOR`)

`[nom variable matériau].setTransDepth([valeurTransDepth])` pour la détermination de la « profondeur » de raytracing (`Depth`), c'est-à-dire le niveau de récursivité des objets transparents multiples

`[nom variable matériau].setFresnelTrans([valeurFresnelTrans])` pour la détermination de la puissance de l'effet Fresnel pour la transparence (`Fresnel`)

`[nom variable matériau].setFresnelTransFac([valeurFresnelTransFac])` pour la détermination du facteur de Fresnel (`Fac`)

Cet onglet propose également, optionnellement, l'usage d'une méthode supplémentaire, pour l'usage de laquelle nous vous renvoyons à la documentation :

`[nom variable matériau].setSpecTransp([valeurSpecTransp)` pour la détermination de la transparence spéculaire (`SpecTra`)

Enfin, trois modes supplémentaires permettent d'activer les boutons poussoirs *OnlyShadow*, *No Mist* et *Env* grâce à la méthode `setMode()` et l'un ou l'autre des arguments suivants : `'OnlyShadow'`, `'NoMist'`, `'Env'`.

Pour reprendre notre exemple de tout à l'heure, le bloc de lignes suivant va nous permettre de donner à notre matériau les reflets qu'il mérite pour ressembler à de l'or digne de ce nom. A noter que pour cette image finale, nous avons dupliqué les sphères faisant usage du matériau ainsi créé, et appliqué tous les modes par défaut de Blender en plus de celui activant le calcul des reflets par raytracing :

```
mat.setRayMirr(0.40)
mat.setMirrDepth(3)
mat.setMode('Traceable','Shadow','Radio','RayMirr')
```

*Fig. 9 : un élevage d'œufs dorés ou de boules de Noël ?*

**Attribution du nouveau matériau à un objet**

Supposons que vous ayez créé un maillage en faisant usage de l'API Python, comme nous l'avions vu dans Lmag #73, et que cet objet est référencé sous la variable Python pyObj (vous pouvez bien sûr avoir spécifié tout autre nom). Supposons également que le matériau que nous créons au cours de cet article soit appelé pyMat. Vous attribuerez le matériau pyMat à l'objet pyObj simplement à l'aide de la ligne suivante :

```
pyObj.materials.append(pyMat)
```

## 2. Création d'une nouvelle texture

La définition d'une texture via l'API Python n'est guère plus difficile que celle d'un matériau, par les mêmes moyens. Il faut cependant au préalable prendre la précaution d'importer depuis le module `Blender` des sous-modules supplémentaires : au minimum `Texture`, et éventuellement `Image` si vous souhaitez faire usage d'images *bitmap* pour habiller vos objets. Le résultat serait donc une ligne d'import ressemblant à :

```
from Blender import Material, Texture, Image
```

Pour créer une nouvelle texture, il suffit de faire appel à la fonction `New()` du module `Texture`, avec le nom Blender de la Texture comme argument de la fonction. Il faut ensuite déterminer le type de texture souhaité, à l'aide de la méthode `setType()`. Parfois, un type de texture admet un sous-type de texture, comme par exemple `Stucci` qui admet les sous-types `Plastic`, `Wall in` et `Wall out`. Le cas échéant, le sous-type est déterminé par l'usage de la méthode `setSType()`. Ensuite, des fonctions optionnelles, dépendant de chaque type et sous-type, doivent être spécifiées pour paramétrer finement les textures.

**Syntaxe élémentaire de la création d'une nouvelle texture :**

```
from Blender import Material, Texture, Image
[nom variable texture] = Texture.New('[nom Blender]')
[nom variable texture].setType('[Type]')
```

`[nom variable texture]` : il s'agit du nom de la variable Python correspondant à la nouvelle texture. Évitez l'usage du point '.' ou des caractères spéciaux au risque de rencontrer des erreurs lors de l'exécution de votre script.

`[nom Blender]` : il s'agit du nom sous lequel la texture apparaîtra dans Blender.

Nous n'explorerons pas tous les types et sous-types possibles, mais nous nous attacherons au moins à décrire deux types de texture : la texture `Image` et une texture procédurale admettant des sous-types : `Stucci`.

Nous devrions ainsi explorer la quasi-totalité des options possibles offertes par les onglets restant à étudier dans les menus *Material buttons* et *Texture buttons*.

### 2.1 La texture Image

Il s'agit d'un grand classique lors de l'usage de Blender, intimement lié aux onglets *Map Input* et *Map To* du menu *Material buttons*. Mais nous allons voir tout cela ; supposons le bout de code suivant :

```
tex = Texture.New('Texture')
tex.setType('Image')
```

Il nous faut maintenant charger une image en mémoire. Cela se réalise aisément au travers d'une méthode issue du sous-module `Image`, qu'il nous faut impérativement avoir importé au préalable :

```
from Blender import Material, Texture, Image
tex.setType('Image')
```

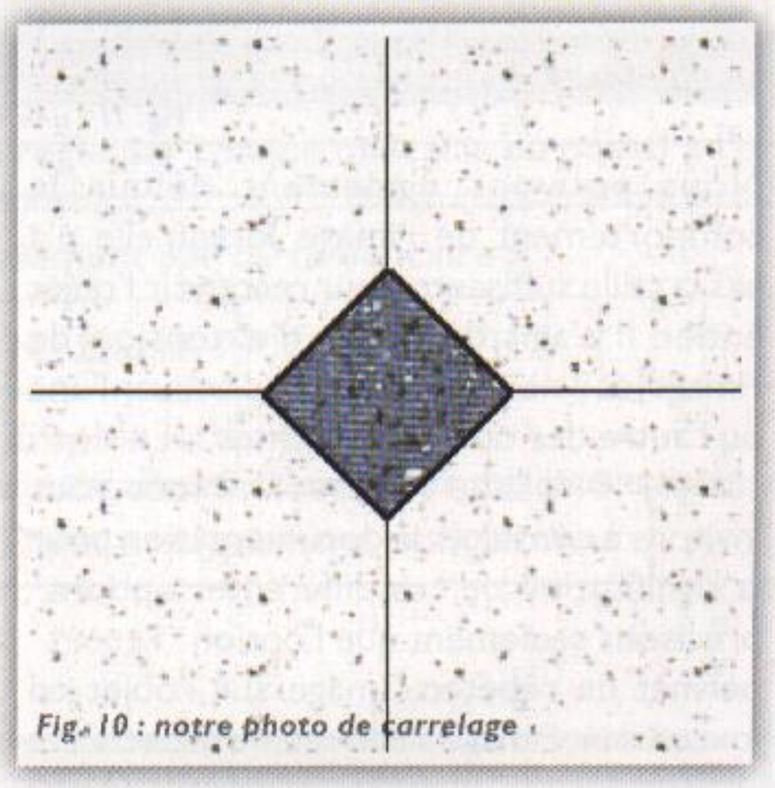


*Fig. 10 : notre photo de carrelage*

Supposons simplement qu'une image intitulée `'carrelage.png'` se trouve dans un certain répertoire, et que vous souhaitiez simplement nommer `'img'` la variable image dans Python. Vous obtenez alors la ligne suivante :

```
img = Image.Load('/chemin/vers/votre/image/carrelage.png')
```

Il ne vous reste alors plus qu'à lier l'image en mémoire à la texture grâce à la commande suivante :

```
tex.image = img
```

Dès lors, il vous est alors possible de spécifier des options, exactement comme sous Blender, ainsi qu'en témoigne la Figure 11 ci-après. En particulier nous pouvons déterminer les options de l'image grâce à la méthode `setFlags()` pour activer certains des boutons de l'onglet Image : `'InterPol'`, `'UseAlpha'`, `'MipMap'`, `'Fields'`, `'Rot90'`, `'CalcAlpha'`, `'StField'`, `'Movie'` et `'Cyclic'`. Comme nous en avons pris l'habitude avec les `Modes`, lors de l'exploration du sous-module `Material`, la casse et la présence des apostrophes simples ' sont toutes deux importantes. Nous aurons donc recours à une ligne de type :

```
tex.setImageFlags('InterPol','MipMap')
```

Vous noterez que toutes les options ne sont pas encore activables au travers de l'API : `NegAlpha` et `Anti` brillent pour l'instant par leur absence.

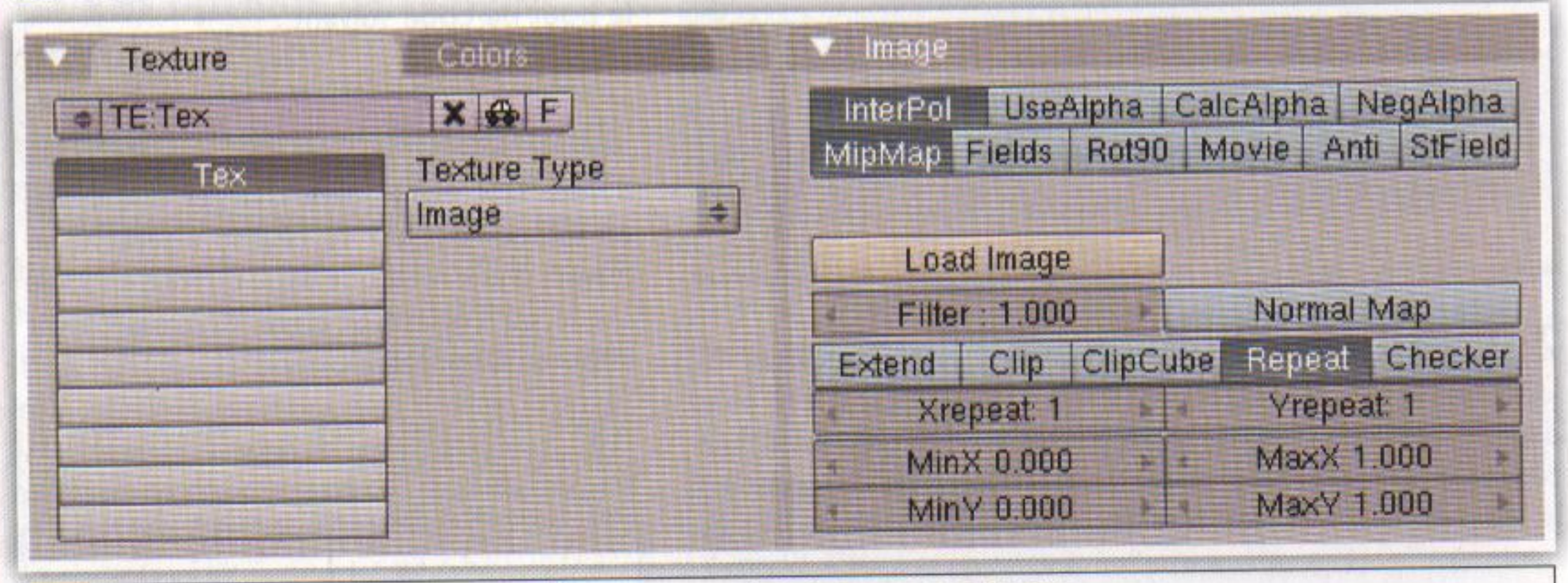



*Fig. 11 : une texture de type Image sous Blender*

Nous pouvons également définir le comportement de l'image lorsqu'elle n'a pas la taille suffisante pour recouvrir l'objet entier. Il s'agit du mode d'extension de l'image, et il nous est possible d'activer l'une ou l'autre des options suivantes : `'Extend'`, `'Clip'`, `'ClipCube'`, `'Repeat'`. Nous vous invitons à consulter la documentation pour la signification de ces différentes options ; précisons seulement que l'option `'Repeat'` permet de répéter l'image sur l'objet en juxtaposant l'image à elle-même dans toutes les directions, tandis que l'option `'Extend'` « étire » les bords de l'image à l'infini pour terminer de couvrir l'objet si l'image n'est pas assez grande pour l'objet. Dans le cadre de notre exemple, nous aurons recours, avec les mêmes contraintes de casse et d'apostrophes que précédemment, à une ligne de type :

```
tex.setExtend('Repeat')
```

S'agissant d'une image, il reste à définir l'origine de la texture par rapport à l'objet (par exemple, par rapport au centre de l'objet, ou par rapport au centre de la fenêtre, etc.) ainsi que le canal de la texture affecté par l'image (par exemple la couleur, la spécularité, la transparence, etc.). Les différentes options accessibles dans Blender sont illustrées sur la figure 12 .

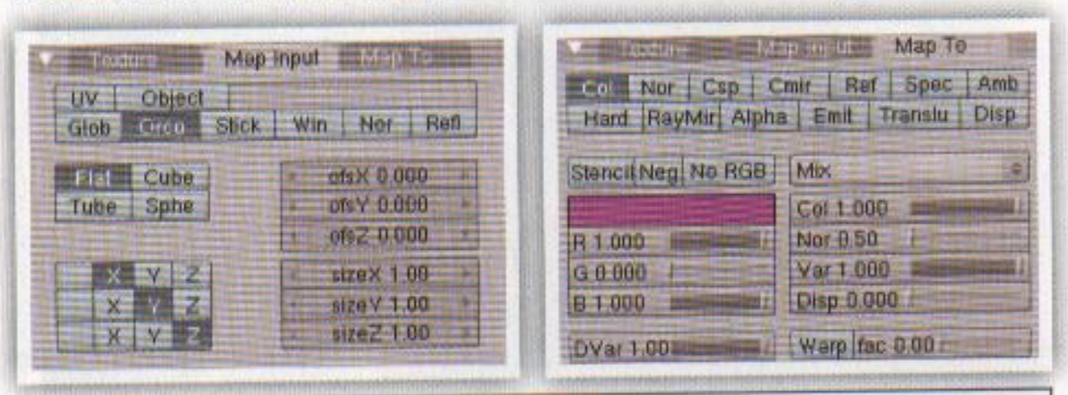



*Fig. 12 : illustration dans Blender des options texco et mapto*

Tout cela se réalisera au travers de la méthode `setTexture()` qui accepte jusqu'à quatre arguments : le premier définit l'index de texture (la valeur peut varier de 0 à 9, sachant que Blender accepte désormais jusqu'à 10 textures au total pour un même matériau, au lieu des 8 accessibles dans le passé) ; le deuxième définit le nom de la texture (précédemment définie) qui est « collée » dans l'indice donné par le premier argument ; le troisième détermine les coordonnées de la texture à employer parmi `ORCO`, `REFL`, `NOR`, `GLOB`, `UV`, `OBJECT`, `WIN`, `VIEW`, et `STICK` (se reporter à la documentation pour l'usage de chacune de ces options), à faire précéder par la chaîne de caractères : `'Texture.TexCo.'` ; enfin, la quatrième et dernière permet de déterminer le canal qui sera « affecté » par la texture : `COL`, `NOR`, `CSP`, `CMIR`, `REF`, `SPEC`, `HARD`, `ALPHA` et `EMIT` (idem, reportez-vous à la documentation) à faire précéder par la chaîne de caractères `'Texture.MapTo.'`. Ainsi, pour associer la texture `tex` à l'indice 0 du matériau `mat`, avec les coordonnées de texture de type `Orco` et une application aux canaux `Col` du matériau, nous aurions une ligne de type :

```
mat.setTexture(0,tex,Texture.TexCo.ORCO, Texture.MapTo.COL)
```

Le résultat est probant, montrant qu'il est également possible de manipuler des textures de type Image au travers de l'API Python de Blender, ainsi qu'en témoigne la figure 13 ci-après.

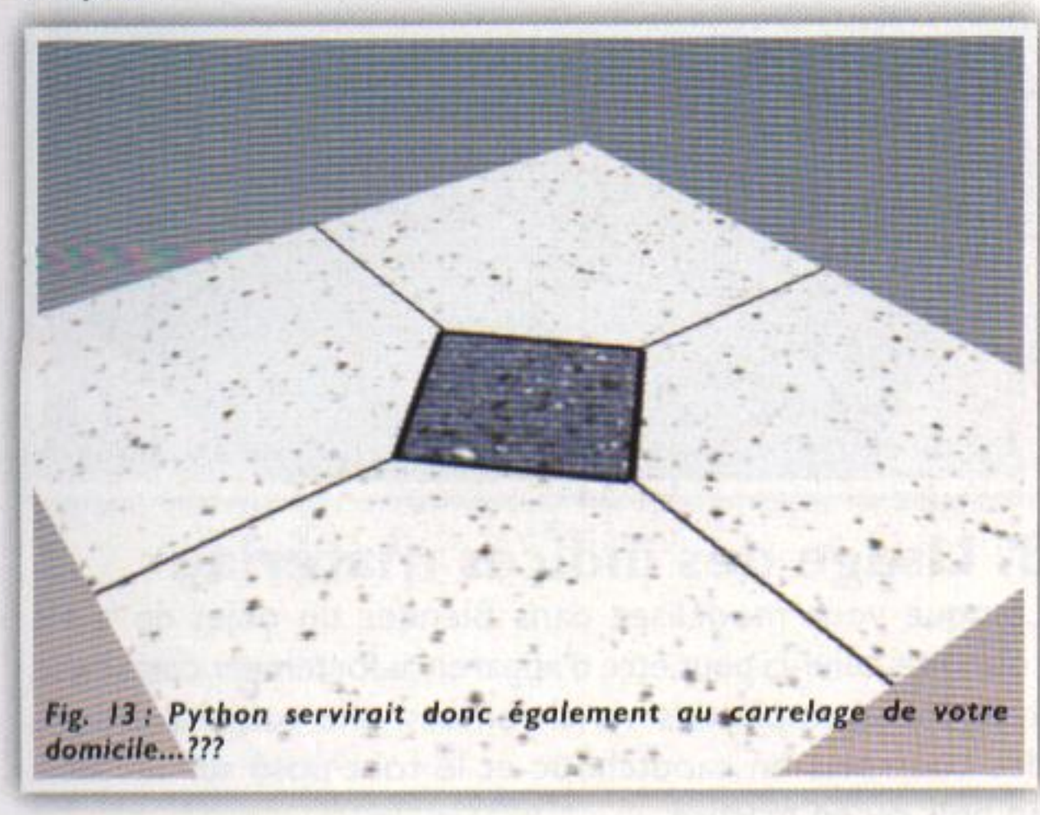


*Fig. 13 : Python servirait donc également au carrelage de votre domicile...???*

Toutefois, il est probable que vous éprouviez le besoin d'activer plusieurs canaux à la fois. Par exemple, Spec et Nor. Malheureusement, la méthode setTexture() n'admet en tout et pour tout qu'un seul mode MapTo et active par défaut le mode Col si rien n'est mentionné.

Il existe toutefois une astuce qui permet d'activer les canaux nominativement. Elle est pleinement explicitée sur l'incontournable site de JM Soler (voir Liens en fin d'article) et s'appuie sur la manipulation des bits. Dans notre cas, nous aurions à insérer ce bout de code pour l'activation des canaux Nor et Spec :

```
mat.setTexture(0,tex,Texture.TexCo.ORCO)
mtextures = mat.getTextures()
        for mtex in mtextures:
                if mtex!=None:
                    mtex.mapto|=Blender.Texture.MapTo['COL']
                    mtex.mapto|=Blender.Texture.MapTo['NOR']
                    mtex.mapto|=Blender.Texture.MapTo['SPEC']
```

Malheureusement, cette opération va fixer le MapTo de toutes les textures à la fois du matériau mat, y compris celles que vous ne souhaiteriez pas. JM Soler propose tout de même une solution palliative qui tend à alourdir le code, mais qui permet néanmoins de régler l'opération en attendant l'amélioration de l'API Python.

A noter que dans Blender, les canaux admettent trois états : activé, désactivé, inversé. Ce tout dernier (qui correspond à ce que vous obtenez sous Blender en cliquant deux fois consécutives sur un bouton de canal, c'est-à-dire le bouton enfoncé mais le texte apparaissant en jaune) n'est pas encore accessible au travers de l'API.

Vous pouvez toutefois activer ou désactiver à volonté les canaux grâce aux opérations booléennes : OU '| = ' pour activer, et ET INVERSE ' & = ~' pour désactiver. Par exemple, pour désactiver le canal COL qui apparaît systématiquement et activer NOR, nous aurions les quelques lignes suivantes en remplacement des précédentes :

```
mat.setTexture(0,tex,Texture.TexCo.ORCO)
mtextures = mat.getTextures()
        for mtex in mtextures:
                if mtex!=None:
                    mtex.mapto&=~Blender.Texture.MapTo['COL']
                    mtex.mapto|=Blender.Texture.MapTo['NOR']
```

**Syntaxe élémentaire de la création d'une texture de type Image :**

```
import Blender
from Blender import Material, Texture, Image
[nom variable image] = Image.Load('[chemin et
nom complet de l'image]')
[nom variable texture].image = [nom variable
image]
[nom variable texture].setImageFlags('[option
1]','[option2]',...)
[nom variable texture].setExtend('[mode
choisi]')
[nom variable matériau].setTexture(0,[nom
variable texture],Texture.TexCo.[mode TexCo],
Texture.MapTo.[mode MapTo])
```

## 2.2 Création d'une texture procédurale

Nous connaissons déjà les bases de la création d'une telle texture, puisqu'elles ne diffèrent pas de celle de la texture Image. A noter toutefois que les deux seuls sous-modules nécessaires sont Material et Texture. La première différence réside dans le fait qu'il nous faut choisir un Type de texture parmi : 'None', 'Clouds', 'Wood', 'Marble', 'Magic', 'Blend', 'Stucci', 'Noise', 'Image', 'Plugin' et 'EnvMap'. Bien sûr, nous avons déjà exploré le cas de l'Image. Nous choisirons donc, à fins d'expérimentation, de nous amuser avec la texture de type 'Stucci'. Malheureusement, ainsi qu'en témoigne la Figure 14, plusieurs types de texture ne sont pour l'instant pas encore accessibles : Distorted Noise, Voronoi, et Musgrave.

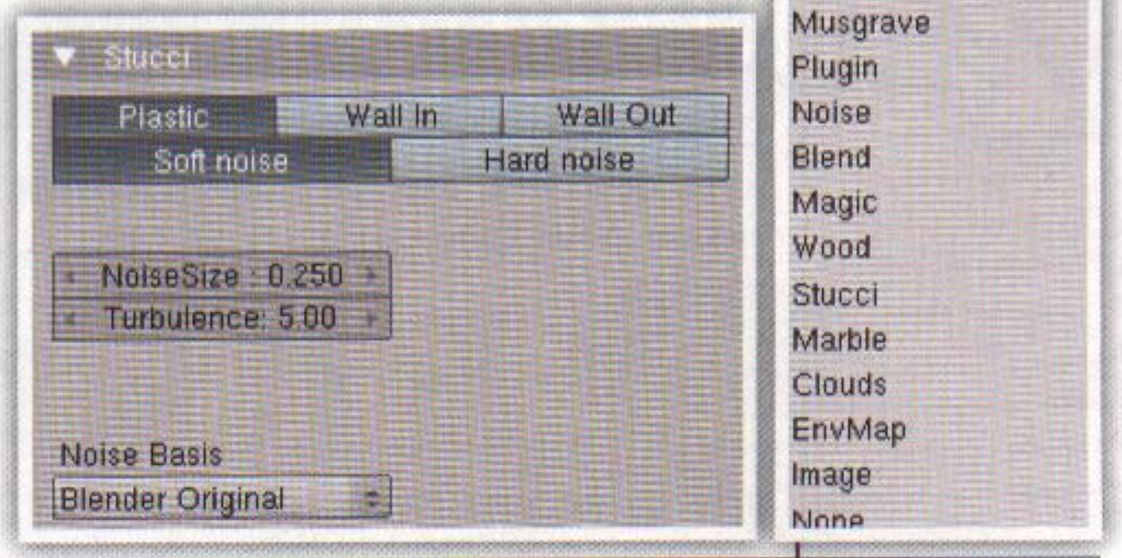



*Fig. 14 : les types de texture accessibles depuis Blender et un zoom sur la texture Stucci*

Pour créer une nouvelle texture, de type Stucci, nous allons commencer simplement par déclarer la nouvelle texture (grâce à la méthode New du module Texture) et ensuite définir son type grâce à la méthode setType(). Uniquement avec ces deux lignes,

nous avons déjà une texture valide, avec tous les paramètres par défaut.

```
tex = Texture.New('Texture')
tex.setType('Stucci')
```

Nous pouvons maintenant affiner le paramétrage de la texture, grâce à la méthode `setSType()` qui permet de définir un sous-type de texture.

Pour une texture de type `Stucci`, nous pouvons accéder aux sous-types suivants : `'StucciPlastic'` pour activer l'option `Plastic`, `'StucciWallIn'` pour `Wall In`, et 'StucciWallOut' pour `Wall Out`. Par exemple:

```
tex.setSType('StucciWallIn')
```

L'observation de la Figure 14 montre également qu'il est possible de choisir entre deux types de *Noise* (Bruit) : `Soft Noise` et `Hard Noise`. Il faut pour cela passer par la variable `noiseType`.

Celle-ci n'admettant pas de « variante » de forme `[var texture].set[méthode]([valeur])`, nous déclarerons donc le type de bruit par une ligne du type `[var texture].[méthode] = [valeur]` :

```
tex.noiseType = 'hard'
```

De la même façon, nous pouvons définir la taille du bruit, c'est-à-dire plus ou moins son facteur de répétition, grâce à la variable `noiseSize`.

Par exemple :

```
tex.noiseSize = 0.55
```

Certaines autres textures procédurales admettent une autre variable, définissant leur niveau de détail : `noiseDepth`.

C'est le cas de `'Clouds'`, `'Marble'`, et `'Magic'`, en particulier. Par exemple, nous aurions pu ajouter pour l'une ou l'autre de celles-ci une ligne :

```
tex.noiseDepth = 3
```

mais ça ne sera pas notre cas pour la texture de type `Stucci`. Nous pouvons donc résumer le code permettant l'obtention de la texture visible sur la Figure 15 par les lignes suivantes :

```
tex = Texture.New('Texture')
tex.setType('Stucci')
tex.setSType('StucciWallIn')
tex.noiseType = 'hard'
tex.noiseSize = 0.55
```

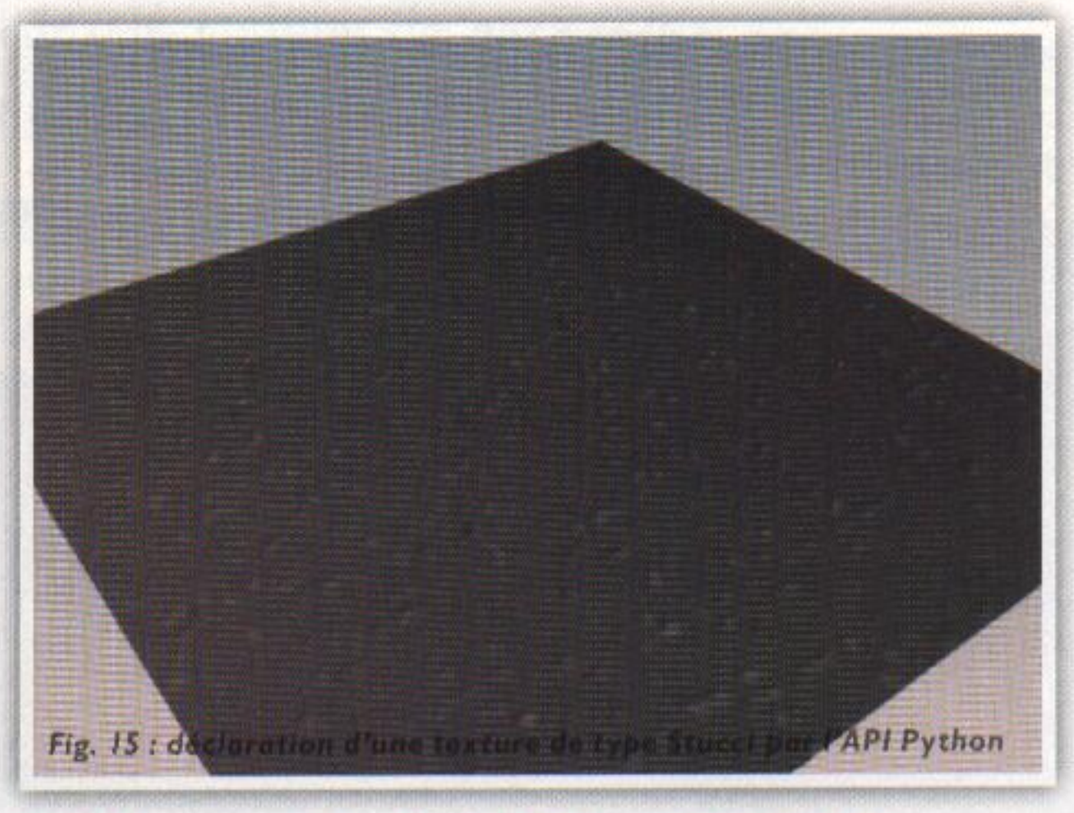


*Fig. 15 : déclaration d'une texture de type Stucci par l'API Python*

## 3. Usage des indices matériau

Lorsque vous modélisez dans Blender un objet de la vie courante, celui-ci peut être d'apparence fortement composite, avec par exemple des vis chromées, une coque plastique, des courroies en caoutchouc et le tout posé sur un socle en bois ou en marbre.

Selon ce qui vous arrange le plus, vous pouvez effectuer votre modélisation en autant que d'éléments que vous le souhaitez, puis « parenter » ([Ctrl]+[P]) le tout à un objet `Empty` pour le manipuler (lors de la composition de la scène ou au cours d'une animation) dans sa globalité.

Ou alors vous pouvez décider de « joindre » ([Ctrl]+[J]) tous ces éléments en un seul maillage unique, bien plus facile à manipuler dans sa globalité.

Vous pourriez dans ce dernier cas craindre de voir tous les matériaux individuels remplacés par un matériau unique et tout votre travail sur les shaders de chaque élément perdu, mais il n'en est rien : Blender est capable de gérer, pour un maillage unique, jusqu'à seize matériaux différents, distinguables les uns des autres par un indice.

L'idée est d'associer à toutes les faces qui ont la même « apparence physique » le même indice de matériau. Au-travers de l'API Python, nous utiliserons donc la méthode `setMaterials()` de la classe `Nmesh` pour déclarer la liste des matériaux accessibles par le maillage, puis enfin la méthode `materialIndex` pour associer à chaque face un indice de matériau.

Dans la pratique, nous allons commencer par lister les matériaux qui vont nous intéresser et les placer dans une variable Python au nom le plus simple et explicite possible, grâce à la méthode `Get()` du sous-module `Material`.

En supposant que nous ayons trois matériaux à récupérer (`'Or'`, `'Carrelage'` et `'Granit'`), cela nous donnerait les lignes suivantes :

```
mat_or = Material.Get('Or')
mat_carrelage = Material.Get('Carrelage')
mat_granit = Material.Get('granit')
```

Supposons que nous souhaitions attribuer ces matériaux au cube que nous avons créé au cours des articles précédents :

```
#### Definition du cube:
# Definition des points de controle:
```

```
liste_des_sommets=[
        [-1, -1, -1,],
        [-1, +1, -1],
        [+1, +1, -1],
        [+1, -1, -1],
        [-1, -1, +1],
        [-1, +1, +1],
        [+1, +1, +1],
        [+1, -1, +1]
]

# Definition des faces:
liste_des_faces=[
        [0,1,2,3],   #face horizontale basse
        [4,5,6,7],   #face horizontale haute
        [0,4,7,3],   #face verticale avant
        [1,2,6,5],   #face verticale arriere
        [0,1,5,4],   #face verticale gauche
        [3,7,6,2]    #face verticale droite
]

CubeMeshData=NMesh.GetRaw()

# Description du cube:
for composante in liste_des_sommets:
        sommet=NMesh.Vert(composante[0], composante[1],
composante[2])
        CubeMeshData.verts.append(sommet)
for face_courante in liste_des_faces:
        face=NMesh.Face()
        for numero_vertex in face_courante:
                face.append(CubeMeshData.verts[numero_vertex])
        CubeMeshData.faces.append(face)
```

Notre `Cube` s'appelle `CubeMeshData`, mais c'est un peu long et mal commode à gérer. Appelons-le plus simplement `m` grâce à la ligne :

```
m = CubeMeshData
```

Les trois matériaux de notre cube ayant déjà été rapatriés sous les variables `mat_granit`, `mat_or` et `mat_carrelage`, nous pouvons tout simplement les attribuer à notre cube grâce à la ligne suivante :

```
m.setMaterials([mat_granit, mat_or, mat_carrelage])
```

L'ordre de déclaration des matériaux dans la liste passée par la méthode `setMaterials()` est important : `Granit` correspond désormais à l'indice 0, `Or` à l'indice 1 et `Carrelage` à l'indice 2.

Nous pouvons désormais attribuer individuellement, à chaque face, un numéro d'indice :

```
m.faces[0].materialIndex = 2
m.faces[1].materialIndex = 2
m.faces[2].materialIndex = 0
m.faces[3].materialIndex = 0
m.faces[4].materialIndex = 1
m.faces[5].materialIndex = 1
```

Enfin, nous pouvons ordonner à l'API Python d'insérer le maillage nouvellement constitué, paré de toutes ses couleurs, dans la scène courante de Blender, et observer le résultat dans la Figure 16.

```
NMesh.PutRaw(CubeMeshData,'Cube',1)
```

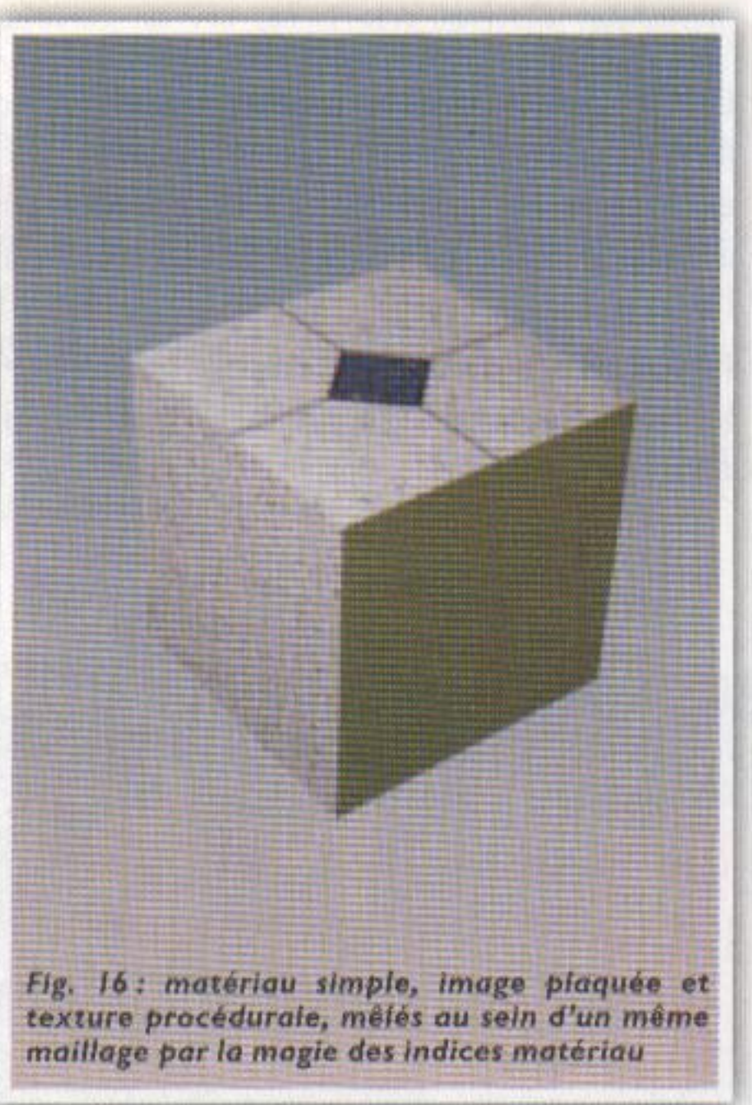


*Fig. 16 : matériau simple, image plaquée et texture procédurale, mêlés au sein d'un même maillage par la magie des indices matériau*

## 4. Conclusion

Vous êtes désormais certainement plus à l'aise avec les notions de matériau, de texture image et de texture procédurale, et vous savez comment les mettre en œuvre au travers de l'API Python. Vous savez également jongler avec les indices matériau et il n'en faut pas beaucoup plus pour faire des choses intéressantes.

Vous avez également noté, une fois de plus, les limites de l'API Python actuelle, mais heureusement vous avez pu voir qu'il reste possible, grâce à une utilisation astucieuse de Python, d'obtenir la plupart des résultats souhaités.

Je profite de ce constat pour rappeler que vous pouvez à tout moment intégrer l'équipe des codeurs de Blender, et ainsi, par exemple, contribuer à améliorer l'API Python jusqu'à ce que son usage soit aussi simple que celui de Blender lui-même.

Pour cet article, mes remerciements vont à Jean-Michel Soler (astuce du MapTo multiple) et à Yves Bailly (débogage de mes scripts et conseil pédagogique sur les indices matériau) pour leur aide discrète mais efficace.

Enfin, une petite note personnelle : j'ai le plaisir de vous annoncer que je suis, depuis le 30 juin dernier, l'heureux papa d'une petite Chloé. Bien évidemment, la maman se porte à merveille même si les nuits ne sont plus aussi tranquilles qu'avant !

## 5. Résumé du code

```
import Blender

from Blender import NMesh, Material, Texture, Image

#### Definition des materiaux:

# Definition du granit:

matGranit = Material.New('Granit')
tex = Texture.New('Granit.Nor')
tex.setType('Stucci')
tex.setSType('StucciWallIn')
tex.noiseType = 'hard'
tex.noiseSize = 0.05
matGranit.setTexture(0,tex,Texture.TexCo.ORCO, Texture.MapTo.NOR)

# Definition de l'or:

matOr = Material.New('Or')
matOr.setRGBCol([1.0,1.0,0.4])
matOr.setSpecCol([1.0,1.0,0.8])
matOr.setMirCol([0.8,0.9,0.7])
matOr.setAlpha(1.0)
matOr.setRef(0.8)
matOr.setSpec(1.4)
matOr.setHardness(20)
matOr.setRayMirr(0.40)
matOr.setMirrDepth(3)
matOr.setMode('Traceable','Shadow','Radio','RayMirr')

# Definition du carrelage:

mat = Material.New('Carrelage')
mat.setSpecCol(1.0,1.0,1.0)
mat.setMirCol(0.9, 0.9, 0.9)
mat.setAlpha(1.0)
mat.setMode('Traceable','Shadow')
tex = Texture.New('Carrelage')
tex.setType('Image')
img = Image.Load('/home/olivier/Documents/Articles/Linux Magazine/
blender-lm-03/files/tile1.jpg')
tex.image = img
tex.setImageFlags('InterPol','MipMap')
tex.setExtend('Repeat')
mat.setTexture(0, tex, Texture.TexCo.ORCO, Texture.MapTo.COL)
```

```
#### Definition du cube:
# Definition des points de controle:
liste_des_sommets=[
          [-1, -1, -1],
          [-1, +1, -1],
          [+1, +1, -1],
          [+1, -1, -1],
          [-1 ,-1, +1],
          [-1, +1, +1],
          [+1, +1, +1],
          [+1, -1, +1]
]
# Definition des faces:
liste_des_faces=[
          [0,1,2,3],   #face horizontale basse
          [4,5,6,7],   #face horizontale haute
          [0,4,7,3],   #face verticale avant
          [1,2,6,5],   #face verticale arriere
          [0,1,5,4],   #face verticale gauche
          [3,7,6,2]    #face verticale droite
]

CubeMeshData=NMesh.GetRaw()

# Description du cube:

for composante in liste_des_sommets:
          sommet=NMesh.Vert(composante[0],
composante[1], composante[2])
          CubeMeshData.verts.append(sommet)
for face_courante in liste_des_faces:
          face=NMesh.Face()
          for numero_vertex in face_courante:
                    face.append(CubeMeshData.
verts[numero_vertex])
          CubeMeshData.faces.append(face)

# Definition des materiaux:

m = CubeMeshData
mat_granit = Material.Get('Granit')
mat_or = Material.Get('Or')
mat_carrelage = Material.Get('Carrelage')
m.setMaterials([mat_granit, mat_or, mat_
carrelage])
m.faces[0].materialIndex = 2
m.faces[1].materialIndex = 2
m.faces[2].materialIndex = 0
m.faces[3].materialIndex = 0
m.faces[4].materialIndex = 1
m.faces[5].materialIndex = 1

NMesh.PutRaw(CubeMeshData,'Cube',1)

Blender.Redraw()
```

*Olivier Saraja,*

*olivier.saraja@linuxgraphic.org*

## LIENS

Le site de développement de Blender : **http://www.blender.org**

La documentation officielle de Python pour Blender : **http://www.blender.org/documentation/237PythonDoc/index.html**

La documentation de Python : **http://www.python.org/doc/2.3.5/lib/lib.html**

Le Site de JM Soler : **http://jmsoler.free.fr/didacticiel/blender/tutor/index.htm**

La page de JM Soler sur l'activation sélective des canaux : **http://jmsoler.free.fr/didacticiel/blender/tutor/cpl_matcanauxmapto.htm**

# ➔ ./configure; make; make install;

Yves Mettier

**EN DEUX MOTS** **Ceux qui ont déjà installé un logiciel en recompilant ses sources ont probablement déjà rencontré la fameuse formule magique *./configure; make; make install*. L'art et la manière de générer le script *configure* qui fait tout a fait l'objet d'un article dans le numéro 24 et est disponible sur internet. Depuis, autoconf a vu son numéro de version passer de 2.13 à 2.5X (2.59 lors de la rédaction de ces lignes) et automake est passé de 1.4 à 1.9. Si la compatibilité antérieure a été souhaitée par les développeurs, ceux-ci ont aussi suggéré de perdre les anciennes habitudes. Cet article a pour objectif de reprendre les choses en montrant, à partir d'un exemple trivial, comment générer les fichiers nécessaires à la commande magique.**

Que sont et que font au juste les auto-tools ? Les auto-tools sont les outils autoconf, automake, libtool et ceux qui gravitent autour (nous évoquerons aclocal plus loin). Au premier abord, ils permettent de générer des fichiers `Makefile` que l'on peut paramétrer avec les options fournies au script `configure`. Si leur intérêt se limitait à cela, les développeurs préféreraient faire leurs propres fichiers `Makefile` et expliquer comment modifier les options par exemple dans un fichier que les `Makefile` pourraient inclure.

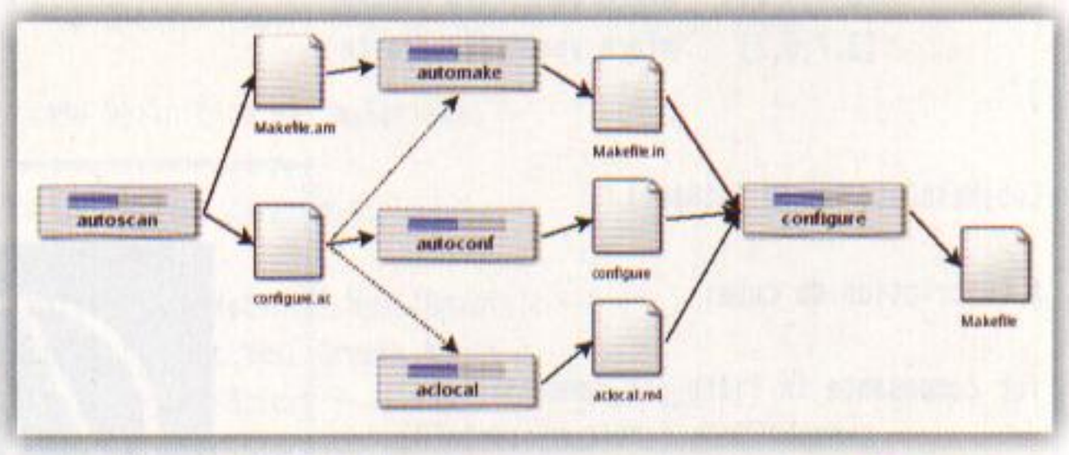
Les auto-tools permettent de générer des fichiers `Makefile` de façon portable, ce qui signifie qu'un programme, tant qu'il ne fait pas appel aux fonctionnalités propres de la plate-forme sur laquelle il doit tourner, pourra être compilé partout où les auto-tools (et les dépendances de votre programme) fonctionnent. Notre exemple, conçu sur Darwin (Mac OS X) fonctionnera aussi bien sur Linux que sur Cygwin (MS Windows). Vous n'avez pas à tenir compte vous-même de la version de `make` et implicitement du format des fichiers `Makefile` et de leurs particularités par exemple.

Mieux, les auto-tools respectent les standards GNU et Unix, en particulier le FHS. Cela fait entre autres la joie des personnes qui créent des paquets pour nos distributions favorites lorsqu'elles respectent aussi le FHS. Ces personnes n'auront aucun mal à adapter les répertoires que votre programme utilise à l'arborescence du système.

Parmi les avantages des auto-tools par rapport à de simples fichiers `Makefile`, nous citerons encore le fait que les cibles (pour make) `clean`, `distcheck` et d'autres sont systématiquement générées, sans le moindre paramétrage supplémentaire de votre part. Il est aussi possible de paramétrer la compilation avec des options `--enable-fonctionnalité` ou `--with-option` pour le script `configure` de notre commande magique. Et j'en passe...

## Aclocal, autoconf et automake

Qui fait quoi ? Nous avons trois commandes et deux types de fichiers (`configure.ac` et les fichiers `Makefile.am`). Au résultat, nous obtenons un fichier `configure` et en l'exécutant, des fichiers `Makefile`. C'est touffu !




Autoconf a pour but de générer un fichier `configure` qui, lorsqu'on le lance, génère des fichiers à partir de fichiers modèles d'extension `.in`. Ainsi, ce script transforme les fichiers `Makefile.in` en fichiers `Makefile`. Voici un mystère d'éclairci. Mais d'où viennent ces fichiers `Makefile.in` ? C'est ici le rôle d'automake. Cet outil prend en compte des fichiers `Makefile.am` et génère des fichiers `Makefile.in` aux mêmes endroits.

Le rôle d'aclocal est de générer un fichier `aclocal.m4` dont nous parlons peu. En effet, il contient des macros (écrites dans le langage m4), dont nous n'avons que rarement besoin de connaître l'existence. Sachez cependant qu'elles sont utilisées par le script `configure` pour effectuer certains tests et définir des variables dont les fichiers `Makefile` auront l'usage. Enfin, le fichier `configure.ac`, fichier de configuration d'autoconf, est utilisé par celui-ci, par aclocal et indirectement par automake. Le premier génère le fichier `configure` en fonction du contenu de `configure.ac`. Le deuxième n'y fait que rechercher les noms des macros afin de copier celles nécessaires dans le fichier `aclocal.m4` à partir de son dépôt (par exemple `/usr/share/aclocal/*`). Le dernier prévoit des variables pour ses fichiers `Makefile.in` en fonction des tests définis dans `configure.ac`. L'ordre d'exécution des commandes est donc aclocal en premier, et ensuite indifféremment autoconf ou automake. Enfin, avant de compiler avec make, lancez le `configure` de la commande magique.

## Notre exemple

Afin d'aborder le sujet, nous allons nous servir d'un exemple simple, un programme qui affiche « Bonjour le monde » et prend fin, dont le code source est réparti sur deux fichiers plus celui des en-têtes. Voici ces trois fichiers :

```c
/* main.c */
#include <stdio.h>
#include <stdlib.h>
#include "afficher.h"
int main(int argc, char**argv) {
        afficher("Bonjour le monde");
        exit(EXIT_SUCCESS);
}
```

```c
/* afficher.c */
#include <stdio.h>
#include <stdlib.h>
#include "afficher.h"
int afficher(char*str) {
        printf("%s\n", str);
        return(0);
}
```

```c
/* afficher.h */
#ifndef AFFICHER_H
#define AFFICHER_H
int afficher(char*str);
#endif
```

Le but de cet article est de compiler le programme `bonjour` à partir de ces sources.

## Répartition des fichiers sources

Tout d'abord, démarrons dans un répertoire vide que nous appellerons `racine` ou `${racine}` dans nos scripts. En général, pour un projet simple, créez un répertoire `${racine}/src/` et mettez-y tous les fichiers sources. Si votre projet doit contenir deux exécutables indépendants, créez un répertoire pour chacun de ces exécutables dans le répertoire `racine`, à la place du répertoire `src/`. Si par contre les deux exécutables partagent certains fichiers sources, vous devez les mettre dans le même répertoire à moins de créer une bibliothèque avec le code commun. Nous voici donc ici avec trois fichiers :

```
$ find .
./src
./src/afficher.c
./src/afficher.h
./src/main.c
```

## Mise en place des fichiers pour autoconf et automake

Vous l'aurez compris, votre projet nécessite un fichier `configure.ac` à la racine et un fichier `Makefile.am` à la racine ainsi que dans tous les sous-répertoires. Commençons par le fichier `configure.ac`. Vous pouvez le générer en lançant la commande `autoscan`. Vous obtiendriez un fichier `configure.scan` à renommer `configure.ac`. Voici ce fichier :

```
#                                               -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.
AC_PREREQ(2.59)
AC_INIT(FULL-PACKAGE-NAME, VERSION, BUG-REPORT-ADDRESS)
AC_CONFIG_SRCDIR([src/afficher.c])
AC_CONFIG_HEADER([config.h])
# Checks for programs.
AC_PROG_CC
# Checks for libraries.
# Checks for header files.
AC_HEADER_STDC
AC_CHECK_HEADERS([stdlib.h])
# Checks for typedefs, structures, and compiler characteristics.
# Checks for library functions.
AC_OUTPUT
```

Ce fichier fait appel à quelques notions que nous verrons dans des articles suivants. Aussi, nous allons nous limiter au fichier `configure.ac` suivant :

```
AC_PREREQ(2.59)
AC_INIT(bonjour, 0.1, ymettier@libertysurf.fr)
AM_INIT_AUTOMAKE
AC_PROG_CC
AC_PROG_MAKE_SET
AC_CONFIG_FILES([
        Makefile
        src/Makefile
        ])
AC_OUTPUT
```

**REMARQUE**

Pour insérer des commentaires, faites-les précéder d'un signe #. Vous pouvez aussi écrire `dnl` à la place, comme cela était le cas avec la version 2.13 d'autoconf.

La première ligne permet d'ôter toute ambiguïté sur la version d'autoconf à utiliser. Cela est utile sur les machines où autoconf-2.5X côtoie encore l'ancienne version 2.13. La ligne `AC_INIT` vous oblige à indiquer le nom de votre projet (notion différente du nom de l'exécutable que nous allons générer), son numéro de version et une adresse électronique que les utilisateurs pourront utiliser pour envoyer des rapports de bogues (pensez à utiliser une liste de diffusion dédiée lorsqu'elle existe). La troisième ligne indique à automake qu'il faut s'initialiser.

Les deux lignes en `AC_PROG_*` testent la présence de programmes et définissent des variables. C'est grâce à elles que le script `configure` va vérifier la présence du compilateur, en général gcc, et définir la variable `$CC`. Idem pour `make` dont la version et l'implémentation (ce n'est pas toujours GNU Make) varient d'une plate-forme à l'autre.

Enfin la ligne `AC_CONFIG_FILES` indique les fichiers à générer (à partir des fichiers portant le même nom, mais avec l'extension supplémentaire `.in`) et `AC_OUTPUT` lance la génération de ces fichiers.

L'étape suivante consiste à éditer des fichiers `Makefile.am`. Dans notre cas d'école, celui de la racine ne contient qu'une ligne définissant les sous-répertoires du projet (mettez une espace en guise de séparateur entre les différents sous-répertoires) :

```
SUBDIRS=src
```

Pour les autres fichiers `Makefile.am`, définissez en premier la variable `bin_PROGRAMS` qui doit contenir le nom des binaires à générer. Notre programme s'appelant `bonjour`, nous y mettons `bonjour`. La variable suivante définit les fichiers sources nécessaires à la compilation du programme `bonjour`. Le nom de la variable est constitué du nom du binaire, suivi de l'extension `_SOURCES`. Le fichiers `${racine}/src/Makefile.am` est le suivant :

```
bin_PROGRAMS=bonjour
bonjour_SOURCES= \
          main.c \
          afficher.c afficher.h
```

Nous verrons dans un article suivant des éléments de syntaxe pour une utilisation un peu plus poussée de ces fichiers `Makefile.am`.

## Génération du script configure

Maintenant que nous avons les fichiers en place, nous pouvons exécuter quelques commandes, dans l'ordre :

```
$ aclocal
$ autoconf
$ automake -a -c
configure.ac: installing `./install-sh'
configure.ac: installing `./missing'
Makefile.am: installing `./INSTALL'
Makefile.am: required file `./NEWS' not found
Makefile.am: required file `./README' not found
Makefile.am: required file `./AUTHORS' not found
Makefile.am: required file `./ChangeLog' not found
Makefile.am: installing `./COPYING'
src/Makefile.am: installing `./depcomp'
```

La première fois que l'on invoque automake de la sorte, nous nous rendons compte qu'il manque certains fichiers. Vous pouvez les créer vides, mais il est préférable de mettre votre nom et/ou celui de l'auteur du logiciel dans le fichier `AUTHORS`. Et quelques lignes dans le fichier `README` ne feront de mal à personne. Pour la postérité, indiquez dans le fichier `ChangeLog` la date de vos manipulations pour ajouter le support d'autoconf/automake ! Lorsque les fichiers manquants sont créés, invoquez `automake -a -c` à nouveau : il ne devrait plus rouspéter. Et vous venez de finir de générer tout le nécessaire à la commande magique que vous pouvez lancer :

```
$ ./configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking for gcc... gcc
[...]
configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating config.h
config.status: executing depfiles commands
$ make
[...]
$ make install
```

## Gérez votre projet

Le projet `bonjour` va probablement prendre de l'ampleur. Voici quelques cas simples et courants que vous pouvez être amenés à rencontrer.

### Ajouter des options au compilateur

Ajouter des options pour modifier le comportement du compilateur ou celui de l'éditeur de liens est chose courante. Vous pouvez ainsi compiler avec l'option `-g` afin de pouvoir déboguer plus facilement avec gdb. Ce genre d'options ne doit pas apparaître dans le projet et c'est à la personne qui compile de les indiquer. Pour cela, elle profite des variables d'environnement `CPPFLAGS` pour le préprocesseur, `CFLAGS` pour le compilateur et `LDFLAGS` pour l'éditeur de liens. Cela donne :

```
$ ./configure
$ make CFLAGS="-O2"
$ make install-strip
```

Nous générons ainsi un exécutable optimisé et l'installons avec la cible `install-strip` qui supprime tout ce qui est inutile dans la table des symboles. Pour en savoir plus sur cette opération, voyez la page de manuel de l'outil strip. Il est néanmoins une option qu'il peut être souhaitable d'indiquer systématiquement au compilateur : l'option `-Wall`. En présence de celle-ci, le compilateur affiche tous les messages d'avertissement qu'il peut (ou presque, voyez la page de manuel de gcc et les options commençant par `-W`). Pour insérer cette option, nous modifions la variable `CFLAGS` via le fichier `configure.ac` et ajoutons ceci à la fin, avant la ligne `AC_CONFIG_FILES` :

```
if test "x$GCC" = "xyes"; then
  CFLAGS="$CFLAGS -Wall"
fi
```

**AA REMARQUE**

Cette opération peut ajouter deux fois ou plus l'option `-Wall`. Mais le compilateur gcc ne vous en tiendra pas rigueur. C'est pourquoi nous ne compliquons pas ces trois lignes avec un code qui teste la présence de l'option dans `CFLAGS`.

### Ajouter ou supprimer un fichier source

Imaginons que nous ajoutions une nouvelle fonctionnalité, par exemple l'appel à la fonction `plouff()` définie dans le fichier `src/plouf.c`. Nous disposons aussi du fichier d'en-têtes `src/plouf.h`. Pour prendre en compte ces deux nouveaux fichiers, éditez le fichier `src/Makefile.am` et ajoutez les noms des deux nouveaux fichiers à ceux existant déjà, avant ou après, peu importe. Voici le nouveau fichier `src/Makefile.am` :

```
bin_PROGRAMS=bonjour
bonjour_SOURCES= \
                 main.c \
                 plouf.c plouf.h \
                 afficher.c afficher.h
```

Comme vous venez de modifier un fichier `Makefile.am`, vous devez relancer automake. Avec les versions récentes d'automake, les fichiers `Makefile` sont générés de telle manière qu'une modification dans un fichier `Makefile.am` est détectée et en exécutant `make`, vous régénérez tous les fichiers impactés par votre modification. La suppression d'un fichier source est similaire à celle d'un ajout : vous modifiez la variable `bonjour_SOURCES` et la suite ne change pas.

## Ajouter le support d'une bibliothèque

Ajouter le support d'une bibliothèque est très simple lorsque celle-ci est placée dans un endroit standard. Il vous suffit d'ajouter une ligne dans le fichier `configure.ac`. Voici celle qui vous permet d'utiliser la bibliothèque mathématique `libm` :

```
AC_SEARCH_LIBS(pow, m)
```

En général, l'absence d'une bibliothèque devrait provoquer l'arrêt du script `configure`. Le troisième argument de `AC_SEARCH_LIBS` est l'action à effectuer si le test est positif et le quatrième celle si le test est négatif. Nous pouvons profiter de ce dernier ainsi :

```
AC_SEARCH_LIBS(pow, m, [], [exit])
```

Vous pouvez aussi afficher un message à la place d'une simple sortie. Pour cela, utilisez `AC_MSG_ERROR`; Voici le résultat pour `libz` :

```
AC_SEARCH_LIBS(gzopen, z, [], [
        AC_MSG_ERROR([zlib est manquante])
])
```

**ATTENTION**

Faites attention ! Une modification de `configure.ac` nécessite la relance d'*aclocal*, d'*autoconf* et d'*automake*.

## Faire le nettoyage

Qu'entendons-nous par nettoyage ? S'il s'agit de supprimer tout ce qui a été compilé, principalement les fichiers objets (dont extension est `.o`), exécutez simplement `make clean`. SI vous voulez par contre supprimer aussi les fichiers générés suite à l'exécution de `configure`, lancez la commande `make distclean`.

Vous pouvez aussi vouloir faire le grand nettoyage et ne garder que les fichiers sources, ainsi que le strict nécessaire à autoconf et automake.

**Dans ce cas, supprimez tout sauf :**

- Les sources de votre code (typiquement les fichiers d'extension `.c` et `.h` ; notez que le fichier `${racine}/config.h` est généré par `configure` : vous pouvez le supprimer aussi) ;
- Les fichiers `Makefile.am` ;
- Le fichier `configure.ac` ;
- Les fichiers `ChangeLog`, `NEWS`, `README` et `AUTHORS` ;
- Si vous les avez modifiés, les fichiers `COPYING` et `INSTALL`.

Après un tel ménage, il ne reste dans notre projet `bonjour` plus que les fichiers suivants :

```
$ find .
.
./AUTHORS
./ChangeLog
./configure.ac
./Makefile.am
./NEWS
./README
./src
./src/afficher.c
./src/afficher.h
./src/main.c
./src/Makefile.am
```

## Générer un paquet bonjour-0.1.tar.gz

Pour générer un tel paquet, nous pensons à la commande `tar` du répertoire `${racine}` renommé au préalable `bonjour-0.1`. Mais il y a beaucoup plus simple : exécutez `make dist`. Vous obtenez votre paquet, avec pour nom et numéro de version ce que vous avez défini dans `configure.ac` sur la ligne `AC_INIT`.

Cela n'est pas encore la meilleure méthode. En effet, le plus propre est de lancer la commande `make distcheck`.

Ainsi, non seulement vous générez le paquet souhaité, mais il est aussi testé pour vous, dans un répertoire différent de celui de développement. Vous mettez ainsi en évidence s'il manque des fichiers, présents dans l'arborescence de développement, mais non déclarés dans les fichiers `Makefile.am`.

**ASTUCE**

Si vous voulez générer un fichier `bonjour-0.1.tar.bz2`, exécutez la commande `make dist-bzip2`. Vous pouvez obtenir les cibles possibles avec un simple `grep "^dist-" Makefile`. Celles d'automake-1.8.5 sont les suivantes : `dist-gzip`, `dist-bzip2`, `dist-tarZ`, `dist-shar` et `dist-zip`.

## Et ce n'est pas fini...

...mais ce sera pour une prochaine fois car il y a tant à dire sur autoconf, automake, et même libtool...

Merci à Guillaume Rousse pour ses précieux conseils lors de sa relecture.

**LIENS**

Le manuel d'autoconf :
**http://www.gnu.org/manual/autoconf/index.html**

Le manuel d'automake :
**http://www.gnu.org/manual/automake/index.html**

*C en action* (O'Reilly).

# → Créer votre logiciel de chat P2P avec Qt

Éric Lacombe

**EN DEUX MOTS** Cet article explique comment créer un logiciel de chat peer2peer en utilisant la bibliothèque Qt. Il s'agit de rendre possible une communication au sein d'un réseau local juste par le fait de lancer l'application. La connexion se fait automatiquement entre tous les pairs du réseau et aucun serveur n'est nécessaire pour s'enregistrer.

Le code du logiciel (Qchat) est écrit en C++ avec la bibliothèque Qt 3. Il est recommandé d'avoir des notions de programmation orientée objet car l'explication du code fourni sera surtout centrée sur la compréhension de l'algorithmique mise en place.

Nous verrons d'abord comment Qchat établit une communication avec tous les pairs présents dans le réseau local (LAN). L'algorithme mis en place permet à toutes les instances de Qchat au sein du réseau de communiquer sans utiliser de serveur central.

De plus, leur comportement pour communiquer est identique. On les considère donc comme des pairs. Par la suite, nous étendrons ses possibilités pour rendre possible la communication entre plusieurs LAN.

Dans ce cas précis, certains hôtes joueront le rôle privilégié de passerelle entre deux LAN (et par conséquent ne seront plus totalement des pairs).

## 1. Le fonctionnement de l'application

Cette partie se focalise sur le fonctionnement de l'application et par conséquent ne traite pas de Qt. Le lecteur n'ayant pas de connaissance sur Qt peut lire cette partie et laisser à plus tard l'explication de l'implémentation.

### 1.1 Tour d'horizon des fonctionnalités

Ce logiciel dans l'état actuel peut communiquer avec toutes les personnes présentes sur un même LAN. La communication se fait de manière standard avec une zone pour visualiser l'ensemble du trafic sur le LAN et une zone pour la saisie des messages.

On trouve également la liste de tous les utilisateurs connectés sur le LAN. La *figure 1* montre l'interface de Qchat.

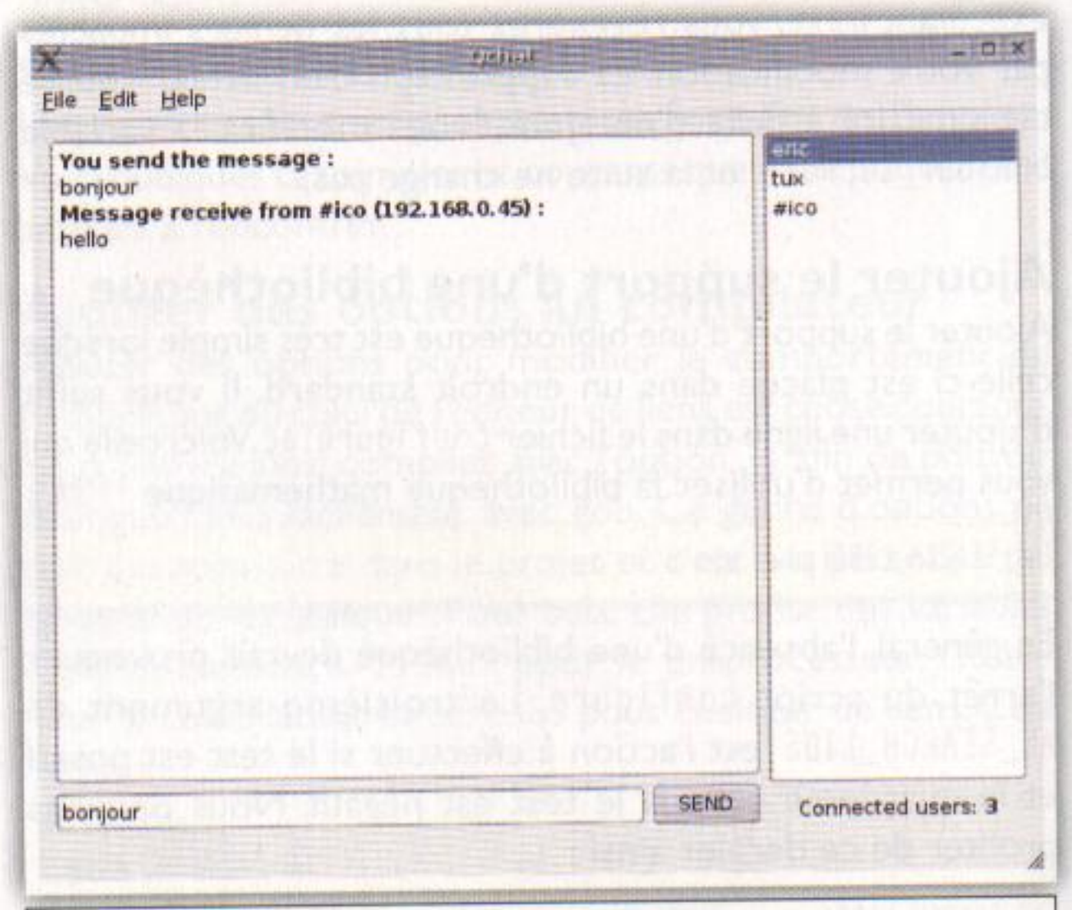



*Fig. 1 : Interface de Qchat*

Il est possible de changer son pseudonyme et d'en répercuter sa visualisation sur l'ensemble des pairs. La dernière fonctionnalité que nous détaillerons et la connexion à des pairs distants pour rendre possible la communication entre différents LAN. Le lecteur pourra implémenter d'autres fonctionnalités s'il le désire. Nous avons placé ce logiciel sous la GNU *Public Licence* et vous trouverez les sources sur le CD du magazine. Avant de commencer à décrire l'algorithme principal de Qchat, notons qu'il est prévu pour une prochaine version d'implémenter le transfert de fichiers entre les pairs du réseau. Au final, l'objectif est de proposer un partage de fichiers entre l'ensemble des utilisateurs connectés sur le réseau.

### 1.2 La communication dans le réseau local

Pour communiquer, les pairs du LAN ont besoin de se reconnaître. Pour cela, nous établissons une connexion en TCP avec tous ceux que l'on trouve sur le réseau local. En fait, les nouveaux arrivants vont émettre un paquet UDP en *broadcast* (i. e. à destination de tout le monde) sur le LAN de manière à avertir tout le monde qu'ils existent.

Décrivons maintenant précisément du point de vue d'un des pairs comment se déroule la procédure. Quand on lance Qchat, les premières actions vont être la création d'une *socket* UDP et TCP (sur un port spécifique identique à tous les pairs du réseau) sur lesquelles on va écouter passivement l'arrivée de paquets. Ce traitement se fait en parallèle de l'exécution du reste du programme. Une fois ces sockets établies, il est temps de communiquer sur le réseau notre identité. Pour cela, nous envoyons d'abord un paquet UDP en broadcast sur le LAN contenant le message `"Bonjour"` pour nous signaler. Supposons le paquet reçu par un autre pair. Sa réaction va être de se connecter à nous en TCP. Notre socket TCP étant créée et *bindée* avant l'émission de notre message `"Bonjour"` aucun problème de synchronisation ne peut se produire. Cette socket que l'on nommera principale est gérée par un

*thread* (fil d'exécution) qui à chaque demande de connexion provenant du réseau local, met en place une autre socket pour communiquer avec le nouvel hôte. Cette dernière socket est stockée dans une table T. La connexion étant établie, chacun envoie un paquet contenant son identité préfixée par un mot-clé signifiant qu'il s'agit d'un message d'identité. Les deux pairs connaissent maintenant leur identité respective et lorsqu'un message de dialogue est envoyé par l'un des deux, l'autre le reçoit et inversement. Notons que l'émission de messages de ce type se produit pour l'ensemble des pairs figurant dans la table T. Nous venons d'expliquer ce qu'entraînait la réception d'un paquet UDP `"Bonjour"` sur un hôte. Étant donné que le paquet est *broadcasté* sur le réseau local, c'est l'ensemble des pairs présents qui va établir une communication avec notre émetteur. La *figure 2* présente le cas d'une émission sur un LAN avec trois hôtes déjà présents. Dans ce contexte, une fois ces connexions établies, le nombre d'hôtes reliés entre eux sera de quatre. Par conséquent lorsqu'un nouveau pair apparaîtra sur le LAN (i. e. émission du paquet `"Bonjour"`), les trois premiers se connecteront à lui ainsi que le petit nouveau de la communauté ;)

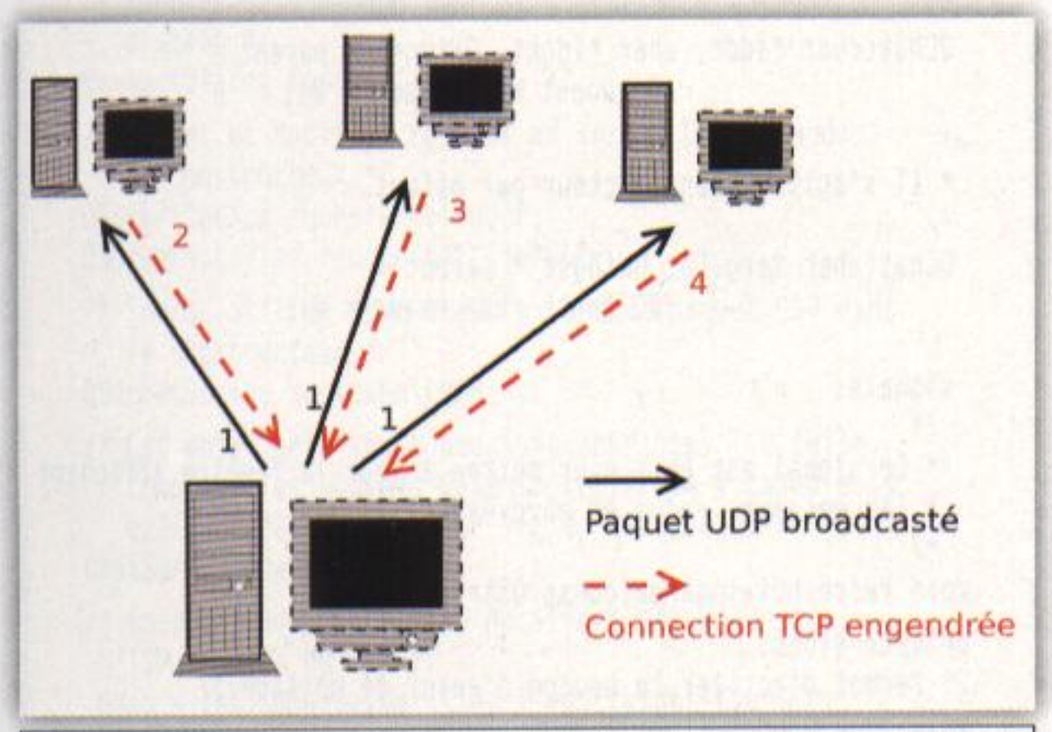



***Fig. 2 : Arrivée d'un nouvel hôte***

Remarquons que même si deux hôtes se connectent en même temps, il n'y a aucun risque que l'un des deux se retrouve isolé, les deux seront reliés avec les hôtes présents sur le LAN. En effet, vu que l'écoute sur les sockets est mise en place avant l'émission du paquet `"Bonjour"`, les deux nouveaux hôtes ne perdront pas trace l'un de l'autre. Cependant, le problème d'une connexion double peut se produire entre ces deux derniers. Dès lors, à chaque émission par l'un de ces deux hôtes de messages de dialogue, l'autre le recevra en double. Ce problème n'est pas géré pour le moment, la recherche d'une solution est laissée en exercice au lecteur.

## 1.3 Discuter avec des utilisateurs distants

Une fonctionnalité supplémentaire permet la communication entre plusieurs LAN. Prenons le cas de deux LAN souhaitant communiquer. Un hôte établit alors une connexion avec un pair extérieur à son LAN en précisant son adresse IP et le port TCP sur lequel l'hôte distant communique. Une fois cette connexion mise en place, l'objectif est de retransmettre tous les messages ainsi que l'identité de l'émetteur d'un LAN sur l'autre et inversement.

Les deux LAN sont donc connectés par l'intermédiaire de deux hôtes qui jouent le rôle important de passerelles. Ces deux hôtes connaissent chacun l'identité de l'autre mais ignorent celles des pairs de l'autre. Pour expliquer sans ambiguïté les différents cas se présentant, appelons nos deux LAN : LAN1 et LAN2 et prenons le point de vue de la passerelle P1 du LAN1. Nous distinguons quatre cas :

- 1. Un message est reçu par P1 provenant du LAN1.
- 2. Un message est reçu par P1 provenant de P2 et initié par P2.
- 3. Un message est reçu par P1 provenant de P2 et initié par un de ses pairs.
- 4. Un message est initié par P1.

Pour le premier cas, P1 effectue en premier lieu les opérations habituelles i. e. l'affichage du message sur l'hôte. Ensuite, il crée un nouveau message pour l'envoyer à P2 (lequel redistribuera le message sur son LAN). Ce message est préfixé par un en-tête `extMsg` lequel signifie au récepteur qu'il s'agit d'un message du LAN1 non initié par P1. Pour que P2 connaisse l'identité de l'émetteur le nom de l'expéditeur est concaténé au corps du message. Dans le deuxième cas, P2 est l'origine du message. P1 va l'afficher et émettre un message modifié sur le LAN1 en agissant comme dans le premier cas (i. e. `message_émis = "extMsg"."identité_de_P2:"."texte"`).

Le troisième cas est plus simple qu'il n'y paraît concernant la réémission. En effet, le message reçu par P1 est déjà façonné comme il le faut (i. e. `message_émis = "extMsg"."identité:"."texte"`). Il n'y a donc qu'à l'afficher et le réexpédier sur le LAN1. Le dernier cas est élémentaire. Le message créé pour le LAN1 est transmis tel quel à P2.

Remarquons que le comportement d'un pair de P1 lors de la réception d'un message extérieur est similaire au troisième cas de P1 concernant l'affichage du message.

Nous venons de présenter comment s'effectuait la communication entre deux LAN avec Qchat. Il est tout à fait envisageable de mettre en relation plusieurs LAN. Deux possibilités s'offrent à nous : soit un seul hôte se connecte aux autres LAN, soit plusieurs hôtes s'y connectent. Des cas problématiques apparaissent. En effet, il est tout à fait envisageable d'être dans une situation où plusieurs connexions sur le même LAN existent. Par conséquent, tous les messages seront dupliqués et donc affichés deux fois sur tous les hôtes.

*TCP : Transmission Control Protocol. RFC 793. TCP est un protocole orienté connexion, c'est-à-dire qu'il permet à deux machines qui communiquent de contrôler l'état de la transmission.*

*UDP : User Datagram Protocol. RFC 768. Protocole très simple sans de contrôle d'erreurs. Il n'est pas orienté connexion... UDP est utilisé avec des services DNS, TFTP, syslog, etc.*

Avant de terminer cette partie, revenons sur les différentes réceptions de P1. Nous avons volontairement omis un cas qui ne se présente pas pour la communication avec seulement deux LAN. Supposons maintenant qu'un LAN3 soit connecté au LAN1 par l'intermédiaire de P1' différent de P1. Dans cette configuration, des messages provenant par exemple du LAN3 vont arriver sur le LAN1 et par conséquent P1 recevra également ce message. Deux comportements sont envisageables : retransmettre ce message (originaire du LAN3) vers le LAN2 ou bien juste l'afficher. Nous avons opté pour la première solution, toutefois il serait bon de mettre en place une option permettant de configurer ce comportement.

# 2. Implémentation

## 2.1 Comment s'organise le code ?

Qchat s'organise en quatre classes principales. La première dont nous allons parler constitue le cœur du logiciel. Il s'agit de la classe `QChat` qui fournit l'IHM principale et contient l'ensemble des propriétés et méthodes nécessaires à son fonctionnement. Cette classe contient un objet `chatServerTCP` (de classe `ChatServer`) dont le rôle est de gérer les connexions entrantes dans le LAN. Les connexions créées par son intermédiaire sont stockées au sein de l'instance de QChat (dans la table de hachage `socketTCP`). Ces deux classes couvrent à elles seules la connexion entre tous les pairs dans un LAN. Les deux prochaines sections expliquent l'implémentation de ces classes en commençant par l'initialisation de l'application (mise en place des sockets réseau). Un autre membre de QChat (`connectTo`) pointe vers un objet de classe `ConnectDialog`. Son rôle est de permettre la création de connexions distantes avec d'autres LAN. Les connexions créées par son biais sont référencées dans une table de hachage (`extSocket`) de la classe `QChat`. C'est dans la section 2.4 que son fonctionnement est dévoilé. Finalement, la dernière classe (`NameDialog`) implémente la fonctionnalité permettant de changer son identité. Sa description constitue la dernière section de notre article.

Avant de regarder le code, signalons que le modelage des interfaces a été fait avec l'application fournie par Trolltech : qtdesigner. Parmi les classes principales dont nous avons parlées, trois proposent une représentation graphique (`QChat`, `ConnectDialog` et `NameDialog`). Le choix, motivé par les recommandations des développeurs de `Qt` [1], s'est porté sur l'utilisation de l'héritage. En effet, chacune de ces classes hérite d'une autre qui constitue son interface graphique et que l'on peut considérer comme une coquille vide car l'ensemble des fonctionnalités attachées sont implémentées dans les classes filles. Le code constituant ces coquilles vides est généré automatiquement par qtdesigner. Nous avons choisi comme convention de nommer ces classes de support graphique par le nom de leur classe fille suivi de la chaîne `"Base"`.

Voyons à présent la description commentée de la classe `Qchat`.

```
class QChat:public QChatBase {
  Q_OBJECT public:
    /* Certaines classes ont besoin d'accéder aux attributs
       de cette classe */
    friend class ChatServer;
    friend class NameDialog;
    friend class ConnectDialog;

    /*
     * Ce constructeur est appelé uniquement pour le mode de
     * débugage consistant à lancer plusieurs QChat sur le
     * même hôte.
     */
     QChat(char *addr, char *ident, QWidget * parent =
                        0, const char *name = 0);

    /*
     * Il s'agit du constructeur par défaut.
     */
     QChat(char *argv[], QWidget * parent =
                        0, const char *name = 0);


 signals:
  /*
   * Ce signal est émis pour mettre à jour la fenêtre affichant
   * les messages reçus et envoyés.
   */
void refreshDialogArea(const QString &);

private slots:
/* Permet d'activer le bouton d'envoi de message */
void enableSendButton();

/* Permet d'envoyer le message courant à l'ensemble des
   connexions actives */
void sendMsg();

/* Fonction de rappel lorsqu'on reçoit un paquet UDP
   signifiant qu'un nouvel hôte du réseau local vient
   de lancer un Qchat */
void dataReceivedUDP();

/* Fonction de rappel associée à la reception d'un
   paquet TCP venant du réseau local */
void dataReceivedTCP(int sockId);

/* Fonction de rappel associée à la reception d'un
   paquet TCP venant d'une connexion à un hôte distant */
void extDataReceived(int sockId);

/* Fonctions de rappel associées aux sous-menus de même
   nom */

void fileExit(); // ferme l'application
void helpAbout();// affiche une boite de dialogue du
                 // type About...

/* Fonction ouvrant une fenêtre de dialogue permettant
   d'éditer son nom et de partager l'information avec
   tous les autres hôtes connectés */
void editName();

/* Fonction ouvrant une fenêtre de dialogue permettant
   de se connecter à un hôte distant et par conséquent
   permet la communication entre différents LAN */
void editConnectTo();
```

```
protected:
  /* On surcharge cette méthode de manière à fermer
     toutes les connexions actives à la sortie du soft */
  void closeEvent(QCloseEvent * event);
  /* Cette fonction met à jour les infos concernant les
     hôtes connectés */
  void refreshInfo();
private:
  /* Cette fonction réalise le travaille du constructeur
     en mettant en place les sockets nécessaires ainsi
     qu'en émettant un paquet en broadcast UDP sur le LAN
     pour avertir de son apparition */
  void constructorWork();
  /* Cet attribut vaut true en mode debug */
  bool debugMode;
  /* Cet attribut permet de considérer toutes les
     connexions TCP comme des connexions distantes
     stockées dans extSocket. Utile uniquement en mode
     debug pour tester le soft sur une seule machine */
  bool connectMode;
  /* Fenêtre de dialogue pour l'édition du nom utilisé
     pour identifier l'hôte */
  NameDialog *askingName;
  /* Fenêtre de dialogue pour se connecter à un hôte
     distant */
  ConnectDialog *connectTo;
  /* Socket et Notifier relatif au socket UDP créé dans
     le constructeur */
  QSocketDevice socketReceiveUDP;
  QSocketNotifier *socketNotifierUDP;
  /* Socket utilisé pour émettre le seul paquet UDP dans
     le constructeur */
  QSocketDevice socketEmitUDP;
  /* Cet objet gère les connexions entrantes TCP (elle
     conserve les sockets dans socketTCP et y associe le
     callback dataReceivedTCP via un Notifier) */
  ChatServer chatServerTCP;
  /* Ensemble des Sockets et Notifiers des connexions
     TCP actives du LAN */
   QMap < int, QSocketNotifier * >socketNotifierTCP;
   QMap < int, QSocketDevice * >socketTCP;
  /* Adresse du LAN */
  QString netAddress;
  /* Variables utilisées uniquement en mode debug */
  int portSource;   // numéro du port d'écoute
  int portDestination;          // numéro du port de
                                // destination
  /* Variable utilisée pour la config du socket UDP en
     mode broadcast */
  int socketOpt;
  /* Vaut true s'il existe au moins un autre pair
     connecté */
  bool existPeer;
  /* Vaut false après l'émission à l'init, en broadcast,
     du paquet UDP signifiant notre présence au LAN. Vaut
     true à la construction pour négliger la réception de
     notre paquet broadcasté. On peut ainsi éviter le
     comportement par défaut qui est de se connecter à
     l'hôte émetteur et ainsi éviter de se connecter à
     nous même */
  bool firstUDP;
  /* Identité de l'utilisateur */
  QString identity;
  /* Noms de l'ensemble des pairs connectés */
   QMap < int, QString > peerIdentity;
  /* Table de correspondances entre les connexions du LAN
     et les entrées dans la liste des hôtes connectés */
   QMap < int, int >socketNumToBoxIndex;
  /* En-tête des types de messages circulant entre tous
     les pairs */
  QString msgHeader;            // message de dialogue
                                // direct
  QString infoHeader;           // message descriptif (nom
                                // d'hôte)
  QString extMsgHeader;         // message de dialogue
                                // indirect
    /* Nombre de pairs (hôtes) connectés */
    int peerNb;
    /* Ensemble des Sockets et Notifiers des connexions
       TCP actives distantes */
     QMap < int, QSocketNotifier * >extNotifier;
     QMap < int, QSocketDevice * >extSocket;
    /* Table de correspondances entre les connexions
       distantes et les entrées dans la liste des hôtes
       connectés */
     QMap < int, int >extSocketToBoxIndex;
};
```

## 2.2 Au démarrage de l'application...

Lorsque l'application se lance la fonction `main` effectue les actions nécessaires au lancement d'une application `Qt`.

```
...
QApplication app(argc, argv);
QChat *chat = new QChat(argv);

app.setMainWidget(chat);
chat->show();

return app.exec();
...
```

L'objet chat est l'instance de la classe principale `QChat`. Sa création est dirigée par son constructeur. Son rôle va être de créer une socket UDP et TCP et d'envoyer le message `"Bonjour"` pour se faire connaître sur le réseau. Notons que la création de la socket TCP est provoquée par l'instance de la classe `chatServerTCP`. Son fonctionnement est expliqué juste après le code du constructeur.

```
QChat::QChat(char *addr, char *ident, QWidget * parent,
             const char *name)
:QChatBase(parent, name), debugMode(false),
connectMode(false),
socketReceiveUDP(QSocketDevice::Datagram),
socketEmitUDP(QSocketDevice::Datagram),
          /* Ici on instancie un objet chatServer qui va se
             charger d'accepter les connexion */
chatServerTCP(PORT_TCP_UDP, this),
netAddress(addr),
portSource(PORT_TCP_UDP),
portDestination(PORT_TCP_UDP),
existPeer(false),
firstUDP(true),
identity(ident),
msgHeader("msg:"),
infoHeader("info:"), extMsgHeader("emsg:")
{
    constructorWork();
}
```

```
void QChat::constructorWork()
{
    peerNb = 1;
    peerBox->changeItem(identity, 0);
    /* Ecoute en UDP sur portSource dans l'atente de
       recevoir des trames de nouveau pair connecté au LAN
     */
    socketReceiveUDP.setBlocking(false);
    socketReceiveUDP.bind(QHostAddress(), portSource);
    socketNotifierUDP =
           new QSocketNotifier(socketReceiveUDP.socket(),
                               QSocketNotifier::Read, this);
    connect(socketNotifierUDP, SIGNAL(activated(int)),
            this, SLOT(dataReceivedUDP()));
    connect(this,
            SIGNAL(refreshDialogArea(const QString &)),
            dialogArea, SLOT(append(const QString &)));
    /* Emission d'un paquet UDP en broadcast pour avertir
       les autres pairs sur le LAN de notre présence */
    int indice = netAddress.findRev(QChar('.'));
    QString bcastAddress = netAddress.left(indice + 1);
    bcastAddress.append("255");
    cout << "Détermination de l'adresse de broadcast : ";
    cout << bcastAddress.append("\n").ascii();
    QByteArray datagram;
    QDataStream out(datagram, IO_WriteOnly);
    out.setVersion(5);
    out << QString("Bonjour");
    QHostAddress bcastAddressHex;
    bcastAddressHex.setAddress(bcastAddress);
    /* on fait appel ici à une fonction C pour que la
       socket accepte le broadscast */
    socketOpt = 1;
    setsockopt(socketEmitUDP.socket(), SOL_SOCKET,
               SO_BROADCAST, &socketOpt, sizeof(int));
    socketEmitUDP.writeBlock(datagram, datagram.size(),
                             bcastAddressHex,
                             portDestination);
}
```

Avant de continuer plus en avant, précisons qu'un mode de debuggage a été prévu pour pouvoir lancer plusieurs applications Qchat sur le même poste.

Cela explique la présence de certains attributs supplémentaires dans la classe `Qchat` comme vous avez pu le constater.

De plus, un autre constructeur est utilisé pour le lancement en mode debug. Nous n'allons cependant pas préciser d'avantage ce mode de fonctionnement de l'application et laissons le lecteur en apprendre plus avec les sources de Qchat, s'il le souhaite.

La classe `chatServerTCP` est construite autour de la classe `QServerSocket` permettant la création d'une socket sur laquelle on attendra des connexions.

Lorsqu'une connexion se produit, la méthode `newConnection` est appelée avec en paramètre la socket à utiliser pour la communication.

La solution utilisée pour différencier une connexion provenant du LAN et une autre extérieure au LAN, suppose que le LAN sur lequel est lancé Qchat utilise des IP de classe C. L'implémentation n'en est que plus simple à comprendre, ce qui constitue un des objectifs de cet article.

```
void ChatServer::newConnection(int socket)
{
    QSocketDevice *socketDevice =
          new QSocketDevice(socket, QSocketDevice::Stream);
    QString addrSrc = socketDevice->address().toString();
    /*
     * On regarde les 3 premiers octets de l'IP
     * ie on fait l'hypothèse qu'il s'agit d'une IP
     * de classe C.
     */
    int cut = parent->netAddress.find(QChar('.'), -2);
    QString cutAddrLAN = parent->netAddress.left(cut);
    QByteArray datagram;
    QDataStream out(datagram, IO_WriteOnly);
    out.setVersion(5);
    out << QString("info:").append(parent->identity);
    /* S'il s'agit d'une connexion du LAN et que le mode
       connect n'est pas actif */
    if (addrSrc.startsWith(cutAddrLAN)) {
        parent->socketTCP[socket] = socketDevice;
        parent->socketTCP[socket]->writeBlock(datagram,
datagram.
size());
        parent->socketNotifierTCP[socket] =
            new QSocketNotifier(socket,
                                QSocketNotifier::Read,
                                this);
        connect(parent->socketNotifierTCP[socket],
                SIGNAL(activated(int)), parent,
                SLOT(dataReceivedTCP(int)));
        /* Il s'agit d'une connexion extérieure au LAN */
    } else {
        parent->extSocket[socket] = socketDevice;
        parent->extSocket[socket]->writeBlock(datagram,
datagram.
size());
        parent->extNotifier[socket] =
            new QSocketNotifier(socket,
                                QSocketNotifier::Read,
                                this);
        connect(parent->extNotifier[socket],
                SIGNAL(activated(int)), parent,
                SLOT(extDataReceived(int)));
    }
    parent->peerNb++;
    if (parent->existPeer == false) {
        parent->existPeer = true;
        connect(parent->sendButton, SIGNAL(clicked()),
                parent, SLOT(sendMsg()));
    }
}
```

Deux tables de hachage sont utilisées en fonction de la provenance de la connexion : `socketTCP` pour les connexions locales et `extSocket` pour les connexions distantes.

Lorsqu'une connexion s'effectue, dans les deux cas, on envoie un message contenant notre identité et on associe un `QSocketNotifier` à la connexion, lequel permet de lier une opération aux réceptions futures de paquets TCP.

## 2.3 Lors de la réception d'un paquet... Quel paquet ?

### 2.3.1 Les paquets UDP

Le premier type de paquet à évoquer est celui des messages "Bonjour" broadcastés sur le LAN en UDP lorsqu'une application Qchat est lancée. A la réception d'un tel paquet, le gestionnaire associé va établir une connexion TCP avec le nouveau pair venant d'apparaître sur le réseau et lui envoyer un paquet TCP contenant notre identité. Voyons à présent le code commenté de ce gestionnaire.

```
/*
 * Cette méthode est le handler associé à la réception d'un paquet UDP
 */
void QChat::dataReceivedUDP()
{
    QString tmp;
    /* on récupère le message du paquet UDP */
    QByteArray datagram(socketReceiveUDP.bytesAvailable());
    socketReceiveUDP.readBlock(datagram.data(),
                               atagram.size());
    QDataStream in(datagram, IO_ReadOnly);
    in.setVersion(5);
    in >> tmp;
    /*
     * Si le qchat est lancé en mode normal, alors on reçoit notre
     * premier paquet UDP que l'on vient de broadcaster => on ne
     * tente donc pas de se connecter à nous même ;)
     * Si le mode debug est actif (i.e. test sur une seule machine
     * avec choix des ports src et dest alors on ne reçoit pas le
     * paquet broadcasté donc on se connecte directement si on
     * reçoit un paquet UDP (venant forcément d'un autre peer).
     */
    if ((firstUDP == false) || debugMode) {
        /* On met en place le calback nécessaire à la
           prochaine communication avec le pair */
        QSocketDevice *socketDevice =
            new QSocketDevice(QSocketDevice::Stream);
        socketTCP[socketDevice->socket()] = socketDevice;
        socketNotifierTCP[socketDevice->socket()] =
            new QSocketNotifier(socketDevice->socket(),
                                QSocketNotifier::Read,
                                this);
        connect(socketNotifierTCP[socketDevice->socket()],
                SIGNAL(activated(int)), this,
                SLOT(dataReceivedTCP(int)));
        /* On se connecte à ce nouveau pair */
        if (socketDevice->
            connect(pairAddress,
                    portDestination) == false) {
            disconnect(socketNotifierTCP
                       [socketDevice->socket()],
                       SIGNAL(activated(int)), this,
                       SLOT(dataReceivedTCP(int)));
            socketTCP.erase(socketDevice->socket());
            socketNotifierTCP.erase(socketDevice->socket());
            cerr << "Problème Huston\n";
        } else {
            /* Si la connexion s'est déroulée sans
               problèmes on envoie un paquet TCP pour
               renseigner de notre identité au pair */
            peerNb++;          // on incrémente le nombre
                               // de connectés sur le LAN
            QByteArray datagram;
            QDataStream out(datagram, IO_WriteOnly);
            out.setVersion(5);
            out << QString("info:").append(identity);
            socketDevice->writeBlock(datagram,
                                     datagram.size());
            /* Au démarrage le bouton SEND n'est pas actif,
               on lui associe un callback dès qu'il y a une
               autre personne sur le réseau */
            if (existPeer == false) {
                existPeer = true;
                connect(sendButton, SIGNAL(clicked()),
                        this, SLOT(sendMsg()));
            }
        }
    } else {
        firstUDP = false;
    }
}
```

### 2.3.2 Les paquets TCP

Nous avons mentionné au cours de la première partie que différents types de messages TCP étaient envoyés entre nos applications Qchat. En effet, nous utilisons trois types de messages :

- Les messages d'informations : ils contiennent pour l'instant seulement l'identité de l'émetteur et sont préfixés par la chaîne contenue dans `QChat.infoHeader`.
- Les messages de dialogue direct : il s'agit des messages de dialogue écrit par des hôtes connectés directement au récepteur (i. e. soit un pair du LAN, soit un hôte privilégié (i. e. une « passerelle ») par lequel un autre LAN est connecté). Ils véhiculent la conversation entre les différents hôtes et sont préfixés par la chaîne contenue dans `QChat.msgHeader`.
- Les messages de dialogue indirect : il s'agit des messages retransmis d'un LAN vers un autre. Ils proviennent d'une « passerelle » qui transmet soit aux hôtes extérieurs auxquels elle est reliée, un message de dialogue direct d'un de ces pairs ; soit aux pairs de son LAN, un message qu'elle a reçu d'un hôte extérieur. Ils véhiculent la conversation entre ces différents hôtes et sont préfixés par la chaîne contenue dans `QChat.extMsgHeader`.

Attardons-nous sur le traitement de ces différents types de messages provenant soit du LAN, soit d'une connexion extérieure.

#### 2.3.2.1 Les messages émis depuis le LAN

On retrouve le traitement de chacun des types de messages dans la méthode `QChat::dataReceivedTCP()` qui est associée à chaque connexion d'un pair du LAN. S'il s'agit d'un message d'information, on met simplement à jour l'identité de l'hôte dans la table `peerIdentity`. Pour les messages de dialogue direct, on les affiche et on les retransmet aux hôtes distants auxquels on est connecté. En ce qui concerne le dernier type de

message, on effectue le même traitement. Cependant comme nous l'avons mentionné dans la première partie, il serait bon de laisser le choix à l'utilisateur quant à la retransmission vers des connexions distantes de paquets de dialogue indirect circulant sur notre LAN. En effet, ce cas se présente par exemple lorsque le LAN 1 est connecté aux LAN 2 et 3 par des passerelles différentes. Par conséquent, des paquets de dialogue provenant du LAN 2 vers le LAN 1 seront des paquets de dialogue indirect au niveau du LAN 1. Quand ces paquets atteindront la passerelle du LAN 1 relié au LAN 3, notre application va les émettre à destination du LAN 3. Bien que ce comportement semble être le plus souvent souhaité, il paraît plus approprié d'en laisser le choix à l'utilisateur. Jetons maintenant un œil au code implémentant notre description.

```
/*
 * Cette méthode est le handler associé à la réception d'un
 * paquet TCP sur le LAN.
 */
void QChat::dataReceivedTCP(int sockId)
{
    if (socketTCP[sockId]->bytesAvailable() > 0) {
          QString msg;
          /* on récupère le message du paquet TCP */
          QByteArray datagram(socketTCP[sockId]->
                              bytesAvailable());
          socketTCP[sockId]->readBlock(datagram.data(),
                                       datagram.size());
          QDataStream in(datagram, IO_ReadOnly);
          in.setVersion(5);
          in >> msg;

          /* Il peut s'agir d'un message d'info d'un pair
             contenant son nom soit parce que la connexion
             viens juste de s'initier, soit parce qu'il vient
             de le modifier */
          if (msg.startsWith(infoHeader)) {

              peerIdentity[sockId] =
                    msg.mid(infoHeader.length(),
                    msg.length() - infoHeader.length());
              refreshInfo();

              /* message de type classique véhiculant les
                 communications des utilisateurs */
          } else if (msg.startsWith(msgHeader)) {

              /* Affichage de l'adresse IP et du message */

              QHostAddress peerAddr =
                    socketTCP[sockId]->peerAddress();

              QString prompt = "<b>Message receive from ";
              prompt.append(peerIdentity[sockId]).
                    append(" (").append(peerAddr.toString()).
                    append(") : </b>");

              msg =
                    msg.mid(msgHeader.length(),
                    msg.length() - msgHeader.length());

              emit refreshDialogArea(prompt);
              emit refreshDialogArea(msg);
              /* traitement pour les connexions distantes :
                 si on est connecté à au moins un utilisateur
                 d'un autre LAN alors il faut jouer le rôle
                 de passerelle et répliquer les paquets que
                 l'on reçoit de notre LAN, avec les
                 modifications adéquates, aux pairs
                 extérieurs */

              if (extSocket.begin() != extSocket.end()) {

                    QByteArray datagram;
                    QDataStream out(datagram, IO_WriteOnly);
                    out.setVersion(5);

                    // on rajoute le nom de l'émetteur
                    msg.prepend(":").
                        prepend(peerIdentity[sockId]);

                    // on rajoute l'en-tête extmsg avant envoi
                    // sur le LAN
                    msg.prepend(extMsgHeader);

                    out << msg;

                    QMap < int,
                        QSocketDevice *
                        >::const_iterator itExt =
                        extSocket.begin();

                    while (itExt != extSocket.end()) {
                        itExt.data()->writeBlock(datagram,
                                          datagram.size());
                        ++itExt;
                    }
              }
              /* Si le message est véhiculé sur le LAN mais
                 provient d'un autre LAN on récupère les
                 informations de l'expéditeur extérieur et on
                 affiche le tout */

          } else if (msg.startsWith(extMsgHeader)) {

              /*
               * On vérifie si on est une passerelle auquel cas
               * on expédie le message pour le propager
               * aux LAN auxquels on est connecté.
               */

              if (extSocket.begin() != extSocket.end()) {

                    QByteArray datagram;
                    QDataStream out(datagram, IO_WriteOnly);
                    out.setVersion(5);

                    out << msg;

                    QMap < int,
                        QSocketDevice *
                        >::const_iterator itExt =
                        extSocket.begin();

                    while (itExt != extSocket.end()) {
                        itExt.data()->writeBlock(datagram,
                                          datagram.size());
                        ++itExt;
                    }
              }


              QString shortMsg;

              // on extrait -> nom:message
```

```
shortMsg =
        msg.mid(extMsgHeader.length(),
                msg.length() -
                extMsgHeader.length());

// on extrait le nom
int cut = shortMsg.find(QChar(':'));
QString extName = shortMsg.left(cut);
extName = extName.right(extName.length() - 1);
extName.prepend("[").append("]");

        // on extrait le message
        shortMsg =
                shortMsg.right(shortMsg.length() - cut - 1);
        QString prompt = "<b>Message receive from ";
        prompt.append(extName).append(" (via ").
                append(peerIdentity[sockId]).
                append(") : </b>");

        emit refreshDialogArea(prompt);
        emit refreshDialogArea(shortMsg);
    }

    /* Dans ce cas, un pair vient de se déconnecter */
} else {
    disconnect(socketNotifierTCP[sockId],
                SIGNAL(activated(int)), this,
                SLOT(dataReceivedTCP(int)));

    peerIdentity.erase(sockId);
    socketTCP.erase(sockId);
    socketNotifierTCP.erase(sockId);

    peerNb--;
    refreshInfo();
}
}
```

#### 2.3.2.2 Les messages émis depuis l'extérieur

Le fonctionnement est similaire à celui décrit précédemment. On retrouve le cas de la réception des messages de dialogue indirect dont la politique de retransmission devrait être paramétrable. Pour l'instant, les messages reçus d'un LAN distant sont réémis sur les autres LAN distants connectés au récepteur. Le code étant très proche du gestionnaire précédent, nous avons choisi de ne pas l'insérer.

#### 2.3.2.3 Remarques

Lorsque plusieurs LAN sont reliés entre eux, les hôtes d'un LAN ne voient pas les hôtes des autres LAN dans la liste des utilisateurs connectés. En effet, cette liste reflète uniquement les connexions effectives entre l'hôte et les autres. Pour remédier à cela, il faudrait mettre en place un protocole plus évolué. Ce protocole mettrait à profit les messages d'information auxquels il rajouterait des renseignements sur l'état des hôtes. Autrement dit, les passerelles informeraient les passerelles des autres LAN des connexions/déconnexions de leurs pairs.

Voyons à présent comment se déroule la connexion entre plusieurs LAN.

## 2.4 Se connecter à un utilisateur extérieur au LAN

La gestion des connexions distantes est faite par la classe `ConnectDialog` que l'on instancie pour chaque nouvelle connexion. La création de ces objets (*figure 3*) passe par le handler `QChat::editConnectTo()` qui est relié à une entrée du menu *edit*.

```
void QChat::editConnectTo()
{
        connectTo = new ConnectDialog(this);
        connectTo->show();
}
```

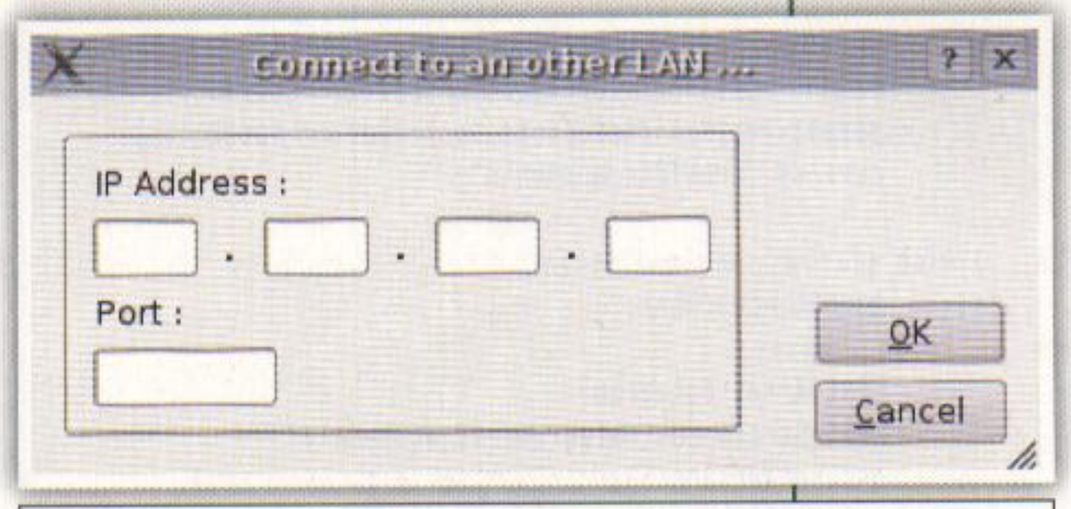



*Fig. 3 : Se connecter à un autre LAN*

Le constructeur de la classe utilise des expressions régulières pour accepter uniquement des adresses IP et ports valides. Voyons à présent la méthode appelée lors de la validation par l'utilisateur de la connexion distante.

Cette méthode crée un `QSocketDevice`, y attache un gestionnaire de réception de paquet i. e. un `QSocketNotifier` (vu en 2.3.2.2), et effectue une connexion avec l'adresse IP et le port donnés par l'utilisateur.

Si la connexion échoue, on libère les ressources, sinon on envoie un message d'information contenant notre identité et on met à jour le nombre d'hôtes connectés. Plus tard, quand l'hôte connecté nous enverra son identité, on mettra à jour notre liste d'utilisateurs. La section suivante précise l'algorithme de gestion de l'affichage des utilisateurs présents.

```
void ConnectDialog::accept()
{
    QString extAddress = firstByte->text().append(".");
    extAddress.append(secondByte->text()).append(".");
    extAddress.append(thirdByte->text()).append(".");
    extAddress.append(fourthByte->text());

    QSocketDevice *socketDevice =
        new QSocketDevice(QSocketDevice::Stream);
parent->extSocket[socketDevice->socket()] =
        socketDevice;

parent->extNotifier[socketDevice->socket()] =
        new QSocketNotifier(socketDevice->socket(),
                                QSocketNotifier::Read, this);

connect(parent->extNotifier[socketDevice->socket()],
            SIGNAL(activated(int)), parent,
            SLOT(extDataReceived(int)));

int port;
```

```
if (portNum->text().isEmpty())
        port = parent->portDestination;
else
        port = portNum->text().toUInt();

if (socketDevice->connect(extAddress, port) == false) {

        disconnect(parent->
                        extNotifier[socketDevice->socket()],
                        SIGNAL(activated(int)), parent,
                        SLOT(extDataReceived(int)));
        parent->extSocket.erase(socketDevice->socket());
        parent->extNotifier.erase(socketDevice->socket());
        cerr << "Problème Huston\n";

} else {
        parent->peerNb++;

        QByteArray datagram;
        QDataStream out(datagram, IO_WriteOnly);
        out.setVersion(5);
        out << QString("info:").append(parent->identity);

        socketDevice->writeBlock(datagram, datagram.size());

        if (parent->existPeer == false) {
            parent->existPeer = true;

            connect(parent->sendButton, SIGNAL(clicked()),
                        parent, SLOT(sendMsg()));
        }
}
    QDialog::accept();
}
```

## 2.5 La visualisation des messages reçus et l'affichage des pairs connectés

### 2.5.1 La visualisation des messages reçus

L'affichage des messages reçus se fait par le mécanisme des *Signals and Slots* de `Qt`. Il s'agit en fait d'associer un gestionnaire (*Slot* dans le jargon de `Qt`) à un signal qui le déclenchera. Dans notre cas, le signal `QChat::refreshDialogArea()` est associé à la méthode `append()` qui permet d'empiler du texte dans le zone `dialogArea`. La mise en relation entre ce signal et ce slot se fait dans le constructeur de QChat par le code suivant.

```
connect(this, SIGNAL(refreshDialogArea(const QString &)),
              dialogArea, SLOT(append(const QString &)));
```

Lors de la réception de messages de dialogue, les gestionnaires que nous avons vus émettent le signal `refreshDialogArea`.

```
...
emit refreshDialogArea(prompt);
emit refreshDialogArea(msg);
...
```

### 2.5.2 L'affichage des utilisateurs connectés

L'affichage des utilisateurs connectés dans la liste (`peerBox`) de droite de Qchat (*figure 1*), se fait par l'intermédiaire de la méthode `QChat::refreshInfo()`.

Cette méthode est appelée lorsque un utilisateur se déconnecte ou lorsque l'on reçoit un paquet d'information contenant soit l'identité d'un nouvel utilisateur, soit une nouvelle identité d'un utilisateur déjà connecté (voir la section 2.6).

Les hôtes qui sont affichés correspondent à tous les pairs du LAN ainsi qu'aux hôtes distants connectés directement à nous.

Il est important de noter que cette méthode effectue à chaque appel une seule action, c'est-à-dire qu'elle va afficher soit un nouvel utilisateur, soit en supprimer un. La présentation qui suit de l'algorithme s'appuie sur ce fait.

L'algorithme utilise une table de correspondances (`socketNumToBoxIndex`) associant un numéro de socket (issue de `socketTCP` ou `extSocket`) à la position que l'hôte associé occupe dans la liste d'affichage.

Le rôle de cet algorithme est de supprimer soit la connexion qui n'est plus valide, soit d'afficher une nouvelle connexion.

Pour cela, la première étape est d'itérer sur la table `socketNumToBoxIndex` pour invalider, s'il y a lieu bien sûr, une entrée qui ne correspond plus à une connexion existante.

Le cas échéant, on sait que la méthode a été appelée pour un utilisateur qui vient de se déconnecter.

Pour supprimer la connexion, on crée un itérateur (`itBox`) sur la table de correspondance, puis à chaque entrée valide, on vérifie si la connexion avec l'utilisateur existe toujours, en vérifiant la présence du numéro de socket (`itBox.key()`) dans `socketTCP` ou `extSocket`.

Quand l'itérateur est sur la connexion qui n'existe plus, on supprime l'utilisateur de l'affichage (i. e. de la liste `peerBox`). Dans le cas où des utilisateurs connectés se trouvaient en dessous dans la liste, on réactualise leur nouvelle position en modifiant la table `socketNumToBoxIndex` via le second itérateur (`itTmp`).

Si aucune entrée n'a été invalidée (`entryDel == false`), on procède alors à la seconde étape car un utilisateur vient de se connecter ou un utilisateur a changé de pseudo.

Pour gérer cela, on itère respectivement sur les tables des connexions existantes `socketTCP` et `extSocket` pour créer, s'il y a lieu, dans la table `socketNumToBoxIndex` une nouvelle entrée. Le cas échéant, il s'agit d'un nouvel utilisateur connecté.

On insère donc une nouvelle entrée (contenant son identité) dans la liste `peerBox` et on met à jour `socketNumToBoxIndex`. Si aucune nouvelle connexion n'est trouvée, on aura parcouru les deux tables `socketTCP` et `extSocket` et effectué la méthode `peerBox->changeItem()` sur l'ensemble des entrée de la liste `peerBox`.

Auquel cas, on aura mis à jour l'identité de l'ensemble des utilisateurs connectés et par conséquent celle de l'utilisateur ayant changé de pseudo.

```
void QChat::refreshInfo()
{
    QString lab = "Connected users: ";
    lab.append(QString("<b>%1</b>").arg(peerNb));

    infoLabel->setText(lab);

    bool entryDel = false;

    QMap < int, int >::const_iterator itBox =
          socketNumToBoxIndex.begin();

    while (itBox != socketNumToBoxIndex.end()) {
        if ((socketTCP.contains(itBox.key()) == false)
            && (extSocket.contains(itBox.key()) == false)) {

            peerBox->removeItem(itBox.data());

            QMap < int, int >::const_iterator itTmp =
                  QMapConstIterator < int, int >(itBox);
            ++itTmp;

            entryDel = true;

            while (itTmp != socketNumToBoxIndex.end()) {

                /* si l'élément supprimé est au-dessus
                   itTmp on met à jour la QMap */
                if (itTmp.data() > itBox.data()) {
                    socketNumToBoxIndex[itTmp.key()] =
                          itTmp.data() - 1;
                }

                ++itTmp;
            }

            socketNumToBoxIndex.erase(itBox.key());
        }

        ++itBox;
    }

    if (!entryDel) {

        QMap < int, QSocketDevice * >::const_iterator it =
            socketTCP.begin();
        QMap < int,
            QSocketDevice * >::const_iterator itExtSock =
            extSocket.begin();

        int boxIndex = 1;

        bool entryModify = false;

        while (!entryModify
               && ((it != socketTCP.end())
                   || (itExtSock != extSocket.end()))) {

            if (it != socketTCP.end()) {

                if (socketNumToBoxIndex.
                    contains(it.key()) == false) {
                    socketNumToBoxIndex[it.key()] =
                          peerNb - 1;
                    peerBox->
                          insertItem(peerIdentity[it.key()],
                                peerNb - 1);

                    entryModify = true;
                } else {
                    peerBox->
                          changeItem(peerIdentity[it.key()],
                                boxIndex);
                }

                ++it;

            } else {

                if (socketNumToBoxIndex.
                    contains(itExtSock.key()) == false) {
                    socketNumToBoxIndex[itExtSock.key()] =
                          peerNb - 1;
                    peerBox->
                          insertItem(peerIdentity
                                [itExtSock.key()],
                                peerNb - 1);

                    entryModify = true;
                } else {
                    peerBox->
                          changeItem(peerIdentity
                                [itExtSock.key()],
                                boxIndex);
                }

                ++itExtSock;
            }

            ++boxIndex;
        }

    }
}
```

## 2.6 Et pour changer son pseudo ?

Voyons finalement la classe responsable du changement de pseudo par un utilisateur déjà connecté.

Comme l'on peut s'en douter, il va s'agir d'envoyer un message d'information contenant notre nouveau pseudo à l'ensemble des hôtes auxquels on est connecté.

Pour cela, la méthode `NameDialog::accept()` associée à la validation de la fenêtre de dialogue de la *figure 4*, effectue l'émission du message aux connexions de la table `socketTCP` et `extSocket`.

Elle modifie également la première entrée de la liste `peerBox` pour mettre à jour notre nouveau pseudo.

Une fois le message reçu par les hôtes auxquels on est connecté, ces derniers vont appeler la méthode `QChat::refreshInfo()` qui rafraîchira (comme nous l'avons vu dans la section précédente) leur liste d'affichage avec notre nouveau pseudo.

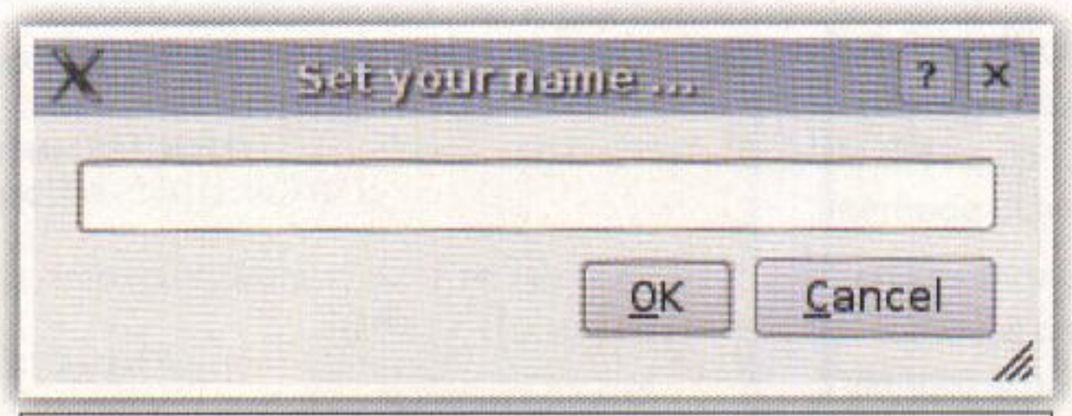



*Fig. 4 : Le changement de pseudo*

d'état. Finalement, le partage de fichiers est la fonctionnalité la plus intéressante restant à implémenter.

Pour le côté pratique, vous trouverez Qchat (placé sous licence GPL) sur le CD du magazine.

```
void NameDialog::accept()
{
        parent->identity = lineEdit->text();

        parent->peerBox->changeItem(parent->identity, 0);

        QByteArray datagram;
        QDataStream out(datagram, IO_WriteOnly);
        out.setVersion(5);
        out << QString(«info:»).append(parent->identity);

        QMap<int, QSocketDevice *>::const_iterator it = parent->socketTCP.begin();

        while (it != parent->socketTCP.end()) {
                it.data()->writeBlock(datagram, datagram.size());
                ++it;
        }

        QMap<int, QSocketDevice *>::const_iterator itExt = parent->extSocket.begin();

        while (itExt != parent->extSocket.end()) {
                itExt.data()->writeBlock(datagram, datagram.size());
                ++itExt;
        }

        QDialog::accept();
}
```

## Conclusion

Nous venons de finir le voyage à travers notre application de chat peer2peer. La création d'un tel logiciel a été facilitée par la puissance d'une bibliothèque telle que Qt.

Nous avons voulu montrer dans cet article qu'il était relativement aisé de construire un tel logiciel.

Bien que de nombreuses fonctionnalités ne soient pas encore implémentées, le lecteur exigeant est encouragé à intégrer les modifications qu'il souhaite :P Rappelons les principaux points sensibles de notre application.

Les connexions doubles entraînant la duplication des messages ne sont pour l'instant pas gérées.

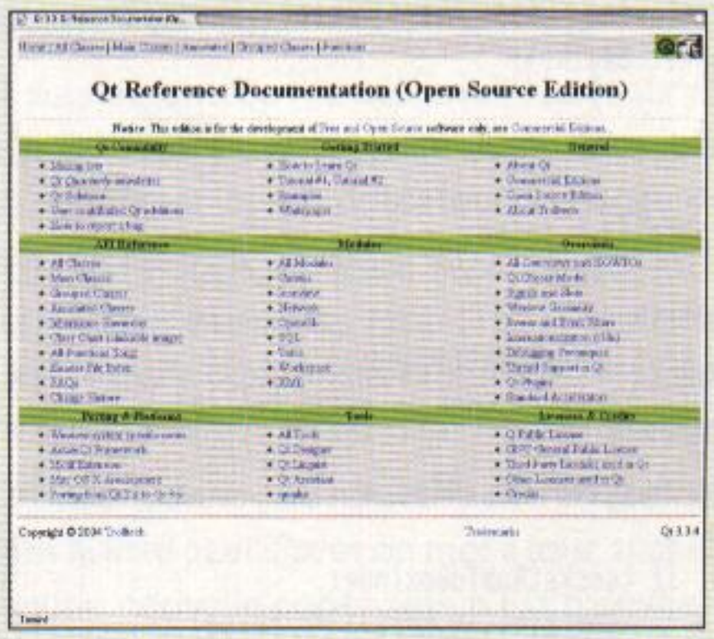
De plus, lorsque plusieurs LAN communiquent via Qchat, les utilisateurs connectés d'un LAN sur l'autre ne sont pas connus (à l'exception des passerelles entre elles) avant réception de messages de dialogue.

Pour obtenir une transparence totale à ce niveau, nous avons donné les idées principales d'un protocole d'échange d'informations

## LIENS

[1] Jasmin Blanchette, Mark Summerfield, *C++ GUI programming with Qt 3*, Prentice Hall PTR.

[2] *Qt Reference Documentation*, Open Source Edition, **http://doc.trolltech.com/3.3/index.html**




[3] Henri Garreta, *Le langage et la bibliothèque C++ Norme ISO*, Ellipses.

**Éric Lacombe,**

**tuxiko@free.fr**

# → Construire des robots pour le web

Philippe « BooK » Bruhat,

**EN DEUX MOTS À la suite de mes articles présentant la librairie LWP dans Linux Magazine 56 à 58, cette nouvelle série décrit l'utilisation de divers outils Perl et techniques d'analyse des sites pour construire des robots qui vont naviguer sur le web à notre place.**

## Des robots ?

On trouve sur le web de nombreux sites d'information, de recherche ou de vente. Le navigateur web et le protocole HTTP ne sont que l'interface permettant d'obtenir l'information souhaitée.

Dans de nombreux cas, on souhaite effectivement accéder à cette information, mais le navigateur web est un obstacle. Il nécessite la présence d'un utilisateur pour entrer l'URL, remplir les formulaires, cliquer sur les liens. Cette série d'articles va vous présenter, par l'exemple, des techniques d'analyse des sites pour construire de petits programmes (des robots !) qui vont, en fonction de divers paramètres, aller chercher l'information seuls, et même réagir et naviguer plus avant *en fonction* de l'information récupérée.

Au fur et à mesure de ces articles, nous allons visiter des sites de plus en plus complexes, construire des robots de plus en plus évolués, pour finir par produire des modules qui nous permettront de réutiliser un moteur de récupération d'information.

## Ma boîte à outils pour le web

### Présentation des outils utilisés

Pour écrire nos robots, nous avons à notre disposition de nombreux outils. Je vous présente ici la liste de ceux que contient ma boîte à outils personnelle :

#### ▶ WWW::Mechanize

Cette librairie Perl encapsule `LWP::UserAgent` pour gérer toute la partie navigation et gestion des formulaires.

Pour une présentation plus complète de `WWW::Mechanize`, je vous renvoie à l'article « LWP, Le Web en Perl (3) », paru dans Linux Magazine 58 et disponible en ligne à l'adresse : **http://articles.mongueurs.net/magazines/linuxmag58.html**.

#### ▶ WWW::Mechanize

se comporte comme un client de haut niveau : il gère par défaut les *cookies*, l'en-tête `Referer`, l'historique des pages visitées, etc. Cela fait autant de code en moins à écrire.

#### ▶ mech-dump

Ce script fourni avec `WWW::Mechanize`, affiche le contenu des formulaires, la liste des images et des liens d'une page web donnée (obtenue avec un simple `GET HTTP`).

Nous verrons qu'il faut s'y prendre autrement quand la page que l'on veut lister n'est pas accessible directement (par exemple, s'il faut une authentification préalable) ou si l'on veut accéder à d'autres informations comme les cookies ou les en-têtes.

#### ▶ HTTP::Proxy

J'utilise un *proxy* maison, qui s'appuie sur mon module `HTTP::Proxy` (disponible sur CPAN), pour extraire les informations intéressantes d'un échange HTTP. Très utile pour espionner l'activité d'un bout de JavaScript ou de Flash.

#### ▶ Mozilla-Firefox

Disposer d'un navigateur couplé à un proxy espion permet de comparer l'activité d'un « vrai » utilisateur avec celle du script en cours d'élaboration.

L'extension *Web Developer* de Firefox permet d'avoir accès aux en-têtes envoyés par le serveur, ainsi que de mettre en valeur certaines parties du HTML. La simple commande `View Source` associée à l'outil de recherche peut s'avérer très utile.

*LiveHTTPheaders* est une autre extension utile pour l'analyse des échanges HTTP : elle permet, dans un onglet de Firefox, de voir les en-têtes des requêtes et réponses voulues (grâce à des expressions régulières de filtrage et d'exclusion) et même de rejouer les requêtes de son choix.

#### ▶ HTML::Parser et HTML::TreeBuilder

La navigation sur certains sites nécessite parfois une analyse poussée du HTML.

L'outil de base pour cette analyse est `HTML::Parser`, un analyseur de HTML par événements. À chaque événement (ouverture/fermeture de balise, etc.), on peut associer un gestionnaire d'événement (*handler*) qui va effectuer un traitement particulier.

Il est souvent plus simple de considérer le document HTML comme un arbre et de naviguer dans les branches de l'arbre. `HTML::TreeBuilder` produit un tel arbre à l'aide de `HTML::Parser` et fournit de nombreuses méthodes de recherche, de navigation et d'élagage dans l'arbre.

#### ▶ ngrep

Parfois, il faut redescendre au plus bas niveau. Dans ces cas-là, ngrep (*network grep*) est vraiment l'outil le plus adapté.

### Détails du proxy espion

Mon seul outil personnel dans la liste présentée précédemment est le proxy basé sur `HTTP::Proxy`. Il est disponible dans la

distribution de HTTP::Proxy, sous le nom « eg/logger.pl ». Le code du proxy est également fourni sur le CD, avec le code des trois scripts de ce mois. Le code de ce proxy n'est pas expliqué en détail ici, car il sort du cadre de cet article. Il s'agit d'un outil d'aide à l'analyse des interactions client/serveur HTTP, dont nous expliquons ci-après l'utilisation.

Pour chaque couple requête/réponse, nous verrons :

- La requête (GET, POST, etc.), et les en-têtes intéressants (par défaut Cookie, Cookie2, Referer, Referrer, Authorization),
- La ligne de statut de la réponse (200 OK, 302 Found, etc.) et les en-têtes intéressants (par défaut Content-Type, Set-Cookie, Set-Cookie2, Location, WWW-Authenticate).
- Les couples clé/valeur passés dans le corps d'une requête POST.

Le proxy accepte comme paramètres en plus des paramètres standards de HTTP::Proxy les paramètres suivants :

- peek <site>

Expression régulière pour déterminer le site à espionner ;

- header <en-tête>

En-tête supplémentaire à afficher (dans la requête et la réponse).

Chaque paramètre peut être répété autant que nécessaire. Si aucun paramètre peek n'est donné, le proxy espionnera tous les sites visités.

À titre d'exemple, voici le résultat de la demande de **http://www.google.com/** par mon client web :

```
$ ./logger.pl peek 'google\.\w+$'
GET http://www.google.com/
302 Found
    Content-Type: text/html
    Set-Cookie: PREF=ID=50559ac18bae0f57:CR=1:TM=1119132978:
LM=1119132978:S=Px8CAVLCC5FoR1NK; expires=Sun, 17-Jan-2038
19:14:07 GMT; path=/; domain=.google.com
    Location: http://www.google.fr/cxfer?c=PREF%3D:
TM%3D1119132978:S%3Dwpjw70CuTrboKsrd&prev=/
GET http://www.google.fr/cxfer?c=PREF%3D:TM%3D1119132978:
S%3Dwpjw70CuTrboKsrd&prev=/
302 Found
    Content-Type: text/html
    Set-Cookie: PREF=ID=e2b4582bd0c2849e:LD=fr:TM=1119132978:
LM=1119132978:S=keTI_KO9ZyhHypD3; expires=Sun, 17-Jan-2038
19:14:07 GMT; path=/; domain=.google.fr
    Location: http://www.google.fr/
GET http://www.google.fr/
    Cookie: PREF=ID=e2b4582bd0c2849e:LD=fr:TM=1119132978:
LM=1119132978:S=keTI_KO9ZyhHypD3
200 OK
    Content-Type: text/html
```

**Voici le résumé de ce que nous observons :**

- La requête vers **http://www.google.com/** provoque l'envoi d'un cookie par le site et une redirection (302) vers **http://www.google.fr/** (mon IP m'a trahi et a permis à Google de deviner que je me connectais depuis la France).

On remarque que le cookie, qui est associé au domaine **google.com** est également passé en paramètre à la requête vers **http://www.google.fr/**.

- **google.fr** renvoie le même cookie que **google.com**, cette fois associé au domaine **google.fr**. Google s'assure ainsi qu'un utilisateur aura le même cookie sur tous ses sites (au moins dans le cas de ce type de redirection géographique).
- Il y a une nouvelle redirection, cette fois vers **http://www.google.fr/**, sans paramètres cachés, puisque le cookie est en place.

Ce proxy nous sera très utile lors de la création de nos robots.

## Remarque sur les sites visités

Tout au long de cette série d'articles, je vais construire des robots pour automatiser la consultation de divers sites que j'utilise couramment. À chaque fois, j'ai choisi des sites qui correspondent à mon utilisation du web et à mes fournisseurs. Il ne s'agit pas de publicité gratuite, juste de la réalité de mon activité sur le web. D'ailleurs, certains n'apprécieraient peut-être pas de savoir que je navigue sur leur site sans voir les pubs de leurs partenaires ou que les cookies qu'ils se donnent tant de mal à m'envoyer sont perdus dès que les scripts se terminent...

Le but de cette série d'articles est de vous permettre, en utilisant les mêmes techniques et outils, de pouvoir à votre tour automatiser votre navigation sur le web selon vos besoins. Nous nous confronterons à des sites de plus en plus complexes au fur et à mesure de notre progression. Maintenant que nous avons nos outils en main, nous allons pouvoir commencer...

## Un script de « copier-coller » (paste)

*Sur IRC, il est assez mal vu de coller sur le canal de larges extraits de code : on inonde le canal pour rien*

Sur IRC, il est assez mal vu de coller sur le canal de larges extraits de code : on inonde le canal pour rien (ce qui est mal en soi) et pour peu que la discussion soit animée, le code va défiler rapidement rendant sa lecture impossible par d'éventuelles bonnes âmes.

La solution a été d'utiliser le web comme presse-papiers mondial : il existe un certain nombre de sites de *paste* (collage) où un *newbie* qui a besoin d'aide peut coller son code (certains sites font même un peu de coloriage de code) et donner l'URL à ceux qui souhaitent l'aider. Une seule ligne est postée sur le canal, et ceux qui veulent donner un coup de main peuvent lire tranquillement le code sur leur navigateur ou le copier/coller (sans les pseudos et l'horodatage de leur client IRC) pour le tester localement.

Il existe même des *paste-bots* qui informent un canal choisi de l'ajout de nouvelles entrées sur le site.

**Voici quelques sites de paste :**

- http://rafb.net/paste/
- http://nopaste.snit.ch:8001/
- http://pastebot.org/
- http://paste.husk.org/

La faiblesse de ces sites, c'est qu'il faut passer par un navigateur web pour publier ces petits bouts de textes, ce qui ralentit énormément le processus.

Par exemple, si j'ai un problème avec un script et que je veux montrer un bout de code sur un canal IRC, je vais devoir ouvrir mon éditeur, copier le contenu du fichier, ouvrir mon navigateur, coller le code (éventuellement en plusieurs fois selon mon éditeur), valider le formulaire, copier l'URL de la page renvoyée en retour et enfin coller cet URL dans le canal IRC.

Un script en ligne de commande prendrait en paramètre le fichier à coller et afficherait l'URL à coller dans le client IRC, tout simplement. J'ai choisi d'automatiser le site http://rafb.net/paste/, car c'est celui que j'utilise le plus souvent.

Le site propose un formulaire avec cinq champs : `Language`, `Nickname`, `Description`, `Convert tabs` et bien sûr une zone de texte pour coller son code. mech-dump va nous donner le nom des champs du formulaire HTML en un tournemain :

```
$ mech-dump http://rafb.net/paste/
POST http://rafb.net/paste/paste.php
  lang=C89            (option) [*C89/C (C89)|C/C (C99)|C++|C#|Jav
a|Pascal|Perl|PHP|PL/I|Python|Ruby|SQL|VB/Visual Basic|Plain Text]
  nick=               (text)
  desc=               (text)
  cvt_tabs=No         (option) [*No|2|3|4|5|6|8]
  text=               (textarea)
  <NONAME>=Paste      (submit)
```

Notre script va commencer par charger les modules requis, puis définir les valeurs par défaut et récupérer les options sur la ligne de commande :

```
#!/usr/bin/perl -w
use strict;
use WWW::Mechanize;
use Getopt::Long;

my %CONF = (
    lang     => 'Plain Text',
    nick     => 'A. Nonyme',
    desc     => '',
    cvt_tabs => 'No',
    text     => '',
);

GetOptions( \%CONF, "lang=s", "nick=s",
"desc=s", "cvt_tabs|tabs=i", "text=s" )
  or die "Bad options";
```

Nous créons ensuite l'objet `WWW::Mechanize`.

**NOTE**

Note : Tout au long de cet article, notre robot sera contenu dans la variable `$m` (pour mech, le surnom donné à `WWW::Mechanize` par son auteur Andy Lester).

Si la description n'est pas fournie dans les options de ligne de commande et qu'un nom de fichier a été passé sur la ligne de commande, c'est le nom du fichier qui sera utilisé. Le texte à coller est récupéré grâce à `<>` qui fait exactement ce qu'on attend : récupérer le contenu des fichiers dont on a passé le nom sur la ligne de commande ou, si rien n'a été passé, il lit le contenu de STDIN (soit la sortie d'un tube, soit ce que l'utilisateur tape sur le terminal).

```
my $m = WWW::Mechanize->new;
$m->get("http://rafb.net/paste/");
die $m->res->status_line unless $m->success;

unless ( $CONF{text} ) {
    $CONF{desc} ||= $ARGV[0];
    $CONF{text} = join "", <>;
}
```

Notez que le script meurt en cas de problème de connexion. L'objet `HTTP::Response` correspondant à la dernière réponse reçue est accessible via la méthode `response()` ou son alias `res()`. Ici, on affiche la ligne de statut en cas d'échec de la requête, c'est-à-dire quand le code de la réponse n'est pas un 2xx ou un 3xx.

D'une manière générale, il est préférable de détecter les problèmes aussi tôt que possible, afin d'éviter que le script « parte en vrille » dès que quelque chose ne se passe pas comme prévu. Pour ce script-ci, cela n'a aucune incidence, mais un script qui automatise des achats ou des transactions importantes a plutôt intérêt à s'arrêter avant de vous endetter pour trois générations ! ;-) (Ceci dit, avec des systèmes comme le *1-Click* d'Amazon, il n'y a plus besoin de donner son numéro de carte bleue pour acheter quelque chose... Autant éviter que notre robot clique n'importe où !)

Il ne reste plus qu'à remplir les champs (avec la méthode `set_fields()`) et valider le formulaire, avant de renvoyer l'URL de la page obtenue.

```
$m->set_fields(%CONF);
$m->submit;
die $m->res->status_line unless $m->success;

print $m->response->request->uri->as_string, "\n";
```

Cette dernière ligne de code mérite quelques explications, car elle est un peu complexe.

Voici ce qu'indique mon proxy personnel au sujet du dernier échange :

```
POST http://rafb.net/paste/paste.php
  Referer: http://rafb.net/paste/
  lang                  => Plain Text
  nick                  => A. Nonyme
  desc                  => test
  cvt_tabs              => No
  text                  => test
302 Found
  Content-Type: text/html; charset=ISO-8859-1
  Set-Cookie: uid=A.+Nonyme; expires=Sat, 25-Jun-05 22:48:16 GMT
```

```
   Location: /paste/results/g9oRiw95.html
GET http://rafb.net/paste/results/g9oRiw95.html
   Cookie: uid=A.+Nonyme
   Cookie2: $Version="1"
   Referer: http://rafb.net/paste/
200 OK
   Content-Type: text/html; charset=ISO-8859-1
```

Lorsque l'on soumet le formulaire, le robot fait un `POST` sur l'URL **http://rafb.net/paste/paste.php**. Le script CGI répond par une réponse `302 Found`, qui indique que le résultat de la requête se trouve temporairement accessible à un autre URL. `WWW::Mechanize` suit automatiquement la redirection. L'URL qui nous intéresse, celui qui mène à la page contenant les données collées, est donc l'URL de la requête faite automatiquement par `WWW::Mechanize`, c'est-à-dire l'URL de la requête associée à la dernière réponse reçue. (L'URL de la redirection annoncé dans la première réponse.)

On remarque au passage que le site note notre *nick* dans un cookie, afin probablement de remplir automatiquement ce champ du formulaire la prochaine fois que nous aurons un paste à faire avec le même navigateur.

En mettant bout à bout les trois blocs de code listés précédemment, nous disposons d'un robot qui sait gérer un site de paste et peut s'utiliser de manière créative :

```
# montrer sa configuration disque
$ df -h | paste-rafb --desc 'df -h'
http://rafb.net/paste/results/YYsdKD11.html

# et son /etc/fstab
$ paste-rafb /etc/fstab
http://rafb.net/paste/results/EFTOEA69.html

# envoyer le code source du script à ses amis (avec coloration
du code)
$ paste-rafb --lang perl `which paste-rafb`
http://rafb.net/paste/results/Nk9csX57.html
```

Comme ce script fonctionne comme un filtre, on peut même l'utiliser depuis vim pour coller du code ou du texte directement depuis son éditeur, avec des commandes comme `:addrw !paste-rafb.addr` est une spécification de lignes à traiter, comme par exemple `%` (tout le fichier), `3,12` (lignes 3 à 12), `+10` (la ligne courante et les 10 suivantes), etc. Attention, il doit y avoir une espace entre le `w` et le `!`, pour que vim comprenne bien que le `!` ne fait pas partie du nom de fichier.

Ce script est particulièrement utile : je m'en sers quasiment tous les jours.

## D'autres outils de paste

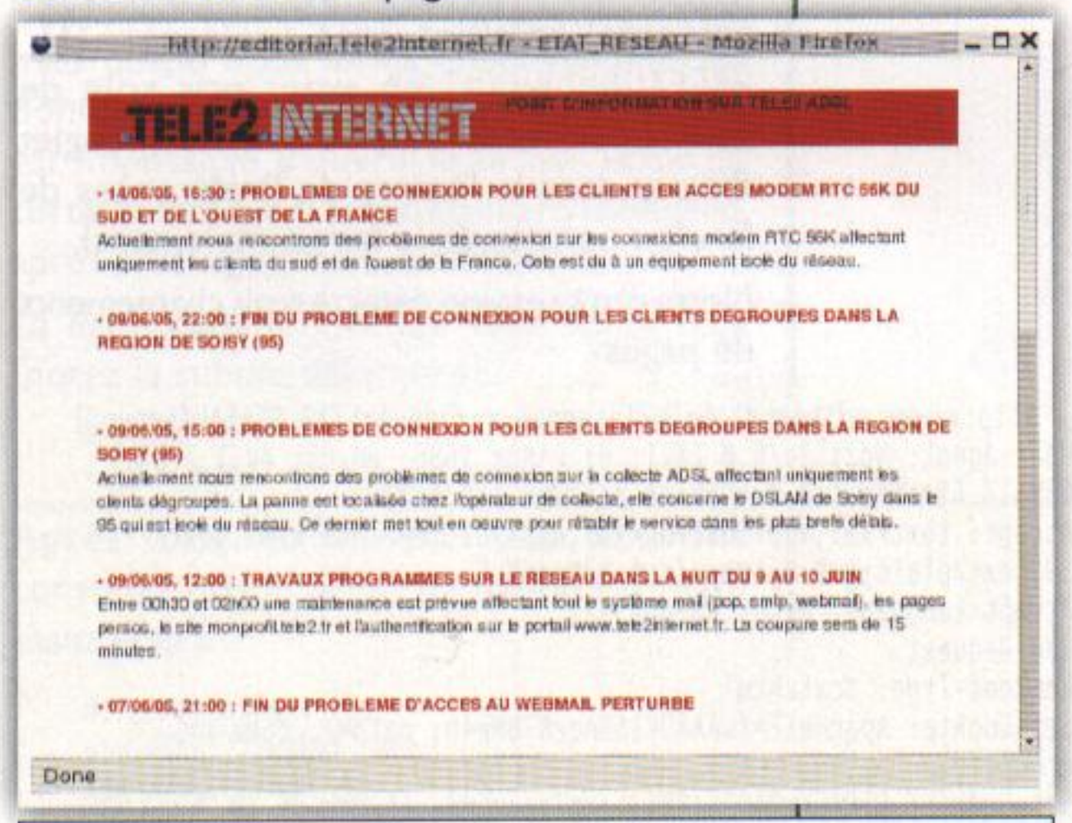
Il existe sur CPAN un module qui s'appelle `WebServices::Paste`. Celui-ci ne me plaît pas trop, car le script fourni doit être modifié à la main pour pointer vers le bon serveur et il n'est pas conçu pour pouvoir être étendu au support d'autres sites de collage.

En revanche, un aspect intéressant de ce module, c'est qu'il utilise un autre module nommé `Clipboard` (sur lequel je n'ai pas eu le temps de me faire une opinion) qui permet de coller le contenu du presse-papiers. Les inconvénients dépassant largement le maigre avantage que cela représente, j'ai préféré passer 10 minutes à écrire mon propre script.

Il serait assez simple de produire un module `WWW::Paste` (par exemple) qui sache gérer les diverses options des multiples sites de collage qui existent (certains ont une option pour qu'un robot annonce le collage sur un canal particulier), et reprendre notre script pour qu'il fonctionne comme un point d'entrée unique pour tous ces sites (avec un fichier `~/.pasterc`, etc.).

## Récupérer une « simple » page d'information

Je suis connecté en ADSL avec Télé2, qui signale sur son site web les problèmes réseau rencontrés. En général, quand je m'inquiète de l'état du réseau, c'est qu'il est déjà coupé pour moi, et je ne peux donc accéder à cette page d'information. L'idéal serait qu'un robot aille récupérer et sauvegarder périodiquement l'information contenue dans cette page.




*État du réseau Télé2*

Le portail **http://www.tele2internet.fr/** a un lien « État du réseau » qui ouvre une *popup* donnant l'état du réseau. Le lien est le suivant :

```
javascript:popup('http://editorial.tele2internet.fr/?page=ETAT_RES
EAU&popup=1','helpPopupWindow',650,450,'scrollbars=1').
```

Donnons la partie utile de ce lien à notre robot :

```
use WWW::Mechanize;
my $m = WWW::Mechanize->new;
$m->get('http://editorial.tele2internet.fr/?page=ETAT_RESEAU&popup=1');
print $m->content;
```

En examinant le contenu du HTML qui nous est renvoyé, nous constatons que le serveur nous répond dédaigneusement :

```
Ce site Web nécessite Netscape 4.x ou une version ultérieure.
```

De plus, si on regarde la ligne de statut de la réponse HTTP :

```
$m->response->status_line;
```

Nous voyons qu'il emploie les grands moyens :

```
400 Bad Request
```

Qu'à cela ne tienne, faisons-nous passer pour Mozilla grâce à la commande suivante :

```
$m->agent_alias("Linux Mozilla");
```

Cette fois-ci, le serveur nous rejette encore avec une réponse `400 Bad Request` et cette phrase :

```
Cette page nécessite Java script.
```

Nous allons continuer notre analyse avec un vrai navigateur. Dans une fenêtre normale, chargeons la page `http://www.editorial.tele2internet.fr/?page=ETAT_RESEAU&popup=1` en ayant pris soin de supprimer les cookies existants (l'onglet *Privacy* dans la fenêtre de Préférences de Firefox permet de le faire simplement).

Notre proxy espion détecte trois chargements de pages :

```
1.    GET http://www.editorial.tele2internet.fr/?page=ETAT_RESEAU&popup=1
       User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.8)
Gecko/20050517 Firefox/1.0.4 (Debian package 1.0.4-2)
       Accept: text/xml,application/xml,application/xhtml+xml,text/
html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
       Accept-Language: en-us,en;q=0.5
    400 Bad Request
       Content-Type: text/html
       Set-Cookie: ApacheEE=fwAAAUK1S3nqeN78Me4h; path=/; domain=.
tele2internet.fr
       Set-Cookie: ETRACK=249904881; expires=Fri, 16-Dec-05 10:39:53 GMT;
path=/; domain=.tele2internet.fr
```

On voit que le serveur répond aussi par un 400 à Firefox, puis lui donne deux cookies.

```
2.    GET ttp://www.editorial.tele2internet.fr/?page=ETAT_RESEAU&popup=1&11
19177593
       Cookie: ApacheEE=fwAAAUK1S3nqeN78Me4h; ETRACK=249904881;
BrowserDetect=passed
       Referer: http://www.editorial.tele2internet.fr/?page=ETAT_
RESEAU&popup=1
       User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.8)
Gecko/20050517 Firefox/1.0.4 (Debian package 1.0.4-2)
       Accept: text/xml,application/xml,application/xhtml+xml,text/
html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
       Accept-Language: en-us,en;q=0.5
    200 OK
       Content-Type: text/html
       Set-Cookie: ETRACK=249904881; expires=Fri, 16-Dec-05 10:39:54 GMT;
path=/; domain=.tele2internet.fr
```

Le JavaScript inclus dans la page a probablement demandé un rechargement. Le numéro passé à la fin de la chaîne de requête (`query string`) correspond à la date au format Unix classique (nombre de secondes depuis janvier 1970).

```
3.    GET http://www.editorial.tele2internet.fr/favicon.ico
       Cookie: ApacheEE=fwAAAUK1S3nqeN78Me4h; ETRACK=249904881;
BrowserDetect=passed
       User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.8)
Gecko/20050517 Firefox/1.0.4 (Debian package 1.0.4-2)
       Accept: image/png,*/*;q=0.5
       Accept-Language: en-us,en;q=0.5
    200 OK
       Content-Type: text/html
       Set-Cookie: ETRACK=249904881; expires=Fri, 16-Dec-05 10:39:55 GMT;
path=/; domain=.tele2internet.fr
```

On remarque qu'à partir de l'étape 2, le client transmet également un cookie `BrowserDetect`. Comme aucun des en-têtes `Set-Cookie` envoyés par le serveur ne correspond, c'est probablement le code JavaScript inclus dans la page qui a fait cet ajout.

Effectivement, le code JavaScript de la deuxième page renvoyée contient la ligne :

```
document.cookie = "BrowserDetect=passed;";
```

Essayons à nouveau en ajoutant simplement le cookie attendu :

```
use WWW::Mechanize;
my $m = WWW::Mechanize->new;
$m->agent_alias("Linux Mozilla");
$m->add_header( Cookie => 'BrowserDetect=passed' );
$m->get('http://editorial.tele2internet.fr/?page=ETAT_
RESEAU&popup=1');
print $m->content;
```

Pour ce cas particulier, nous ajoutons le cookie avec la méthode `add_header()` de `WWW::Mechanize`. Attention cependant, cette méthode ajoute l'en-tête en question à **chacune des requêtes** qui sera effectuée par le robot (l'en-tête en question est stocké dans la liste des « en-têtes spéciaux » du robot). Si l'en-tête ne devait être ajouté que pour cette requête-ci, nous pourrions le supprimer ensuite avec `delete_header()`.

En implantant nous-mêmes ce cookie, nous avons réussi à faire croire au serveur de Télé2 que le code JavaScript a reconnu le navigateur attendu. Et nous capturons ainsi la page d'information.

Il ne reste plus qu'à en extraire les informations utiles. Voici un extrait de la page en question :

```
<!--début de la zone à copier-coller -->
<span class="apptextBold"><font color="#CC0000">
<!--début titre : bien modifier date et heure-->
 14/06/05, 16:30 : PROBLEMES DE CONNEXION POUR LES CLIENTS EN
ACCES MODEM RTC 56K DU SUD ET DE L'OUEST DE LA FRANCE<!--fin titre
-->
</font></span>
<span class="apptextPlain"><br>
<!--début texte -->
Actuellement nous rencontrons des problèmes de connexion sur
les connexions modem RTC 56K affectant uniquement les clients du
sud et de l'ouest de la France. Cela est du à un équipement isolé
du réseau.<!--fin texte -->
</span><BR><BR>
<!--fin de la zone à copier-coller -->
```

Outre les commentaires destinés aux opérateurs mettant les informations à jour, on voit du HTML qui sert à structurer le texte dans des blocs `<span>` selon les classes `apptextBold`

pour les titres et apptextPlain pour les explications associées. Nous allons utiliser HTML::TreeBuilder pour récupérer le texte en question.

```
my $tree =
  HTML::TreeBuilder->new_from_content( $m->content );
print
  $_->as_trimmed_text,
  $_->attr('class') eq 'apptextBold' ? "\n" : "\n\n"
  for $tree->look_down( _tag => 'span', class => qr/apptext/
);
```

La première ligne crée un objet HTML::Tree à partir du contenu de la réponse téléchargée par notre objet WWW::Mechanize. La méthode look_down() renvoie la liste des branches de l'arbre HTML correspondant à la requête. Dans notre cas, nous voulons les nœuds ayant pour balise  et pour attribut class une valeur qui corresponde à l'expression régulière qr/apptext/. On affiche ensuite le texte des éléments avec la méthode as_trimmed_text() de chacun des objets HTML::Element renvoyés. Celle-ci se comporte comme as_text(), en supprimant les blancs en début et fin de ligne. On saute une ligne après le titre et deux lignes après le contenu.

La visualisation du résultat montre que les opérateurs de Télé2 ne savent pas fermer correctement leurs balises . Nous ne pouvons donc même pas compter sur les styles pour faire un affichage correct !

Heureusement, le texte contient des puces (caractère \x95) avant chaque nouveau titre. Il est probable que les opérateurs n'oublieront pas celui-ci. Nous allons donc nous appuyer dessus pour faire notre découpage :

```
my $tree = HTML::TreeBuilder->new_from_content( $m->content );
print
  map { s/\n*\x95/\n\n*/g; $_ }
  map { $_->as_trimmed_text . "\n" }
  $tree->look_down( _tag => 'span', class => qr/apptext/ );
```

Cette fois-ci, nous obtenons le résultat souhaité :

```
   * 14/06/05, 16:30 : PROBLEMES DE CONNEXION POUR LES CLIENTS EN
ACCES MODEM RTC 56K DU SUD ET DE L'OUEST DE LA FRANCE
   Actuellement nous rencontrons des problèmes de connexion sur
les connexions modem RTC 56K affectant uniquement les clients du
sud et de l'ouest de la France. Cela est du à un équipement isolé
du réseau.

   * 09/06/05, 22:00 : FIN DU PROBLEME DE CONNEXION POUR LES
CLIENTS DEGROUPES DANS LA REGION DE SOISY (95)
```

J'ai donc mis ce script dans ma *crontab* :

```
# infos Telé2
*/10 * * * * ~/bin/tele2 > ~/tele2
```

À la coupure suivante de ma connexion, j'ai pu constater mon erreur : quand le réseau est indisponible, le script ne renvoie rien mais écrase quand même le fichier d'information... Autrement dit, je perds l'information en cas de coupure du réseau, exactement quand j'en ai besoin !

La première chose à faire, c'est de sortir du programme en cas d'erreur :

```
# juste après le $m->get(...);
exit unless $m->success;
```

La méthode success() renvoie une valeur booléenne indiquant si la réponse est un succès (code 2xx). Cette méthode encapsule un appel à $m->response->is_success(). Ce qui pose problème en réalité, c'est la redirection de la sortie standard par le *shell*, qui écrase le fichier destination même si le script ne produit aucune sortie. On ne peut pas se contenter d'ajouter en fin de fichier (append) avec le >> du shell, car on ajouterait l'intégralité de la page d'information à notre fichier à chaque lancement du script !

C'est donc à notre script lui-même de gérer la sauvegarde dans un fichier. J'ai choisi le comportement suivant : le premier paramètre sur la ligne de commande sera le nom du fichier dans lequel sauvegarder l'information, sinon la sortie se fait sur STDOUT comme précédemment.

La sélection de la sortie se fait facilement, en réouvrant STDOUT

```
# sélection de la sortie
if (@ARGV) {
    open STDOUT, ">", $ARGV[0]
      or warn "Impossible de rediriger STDOUT sur $ARGV[0]: $!\n";
}
```

C'est-à-dire que nous faisons l'équivalent de > fichier, non pas au démarrage du script, mais quand nous avons effectivement des données à sauvegarder dans ce fichier. S'il est impossible d'ouvrir le fichier voulu, la sortie se fera sur la sortie standard habituelle, après affichage d'un avertissement.

La ligne dans la crontab sera désormais (notez la subtile différence) :

```
# infos Telé2
*/10 * * * * ~/bin/tele2 ~/tele2
```

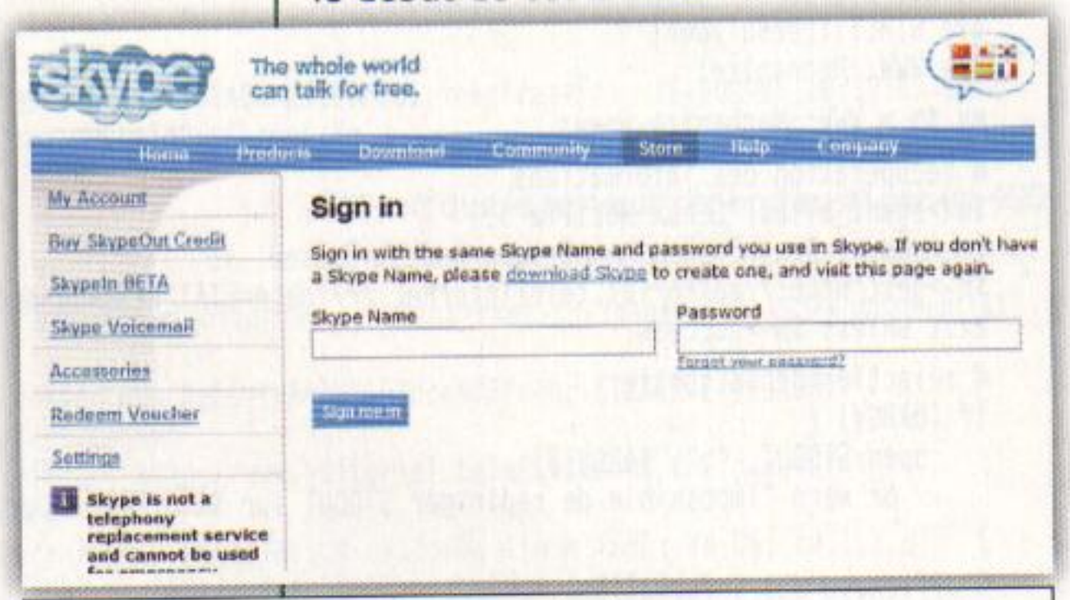
Après tous ces efforts, nous pouvons constater que le script lui-même est resté assez court :

```
#!/usr/bin/perl
use HTML::TreeBuilder;
use WWW::Mechanize;

my $m = WWW::Mechanize->new;

# récupération des informations
$m->agent_alias("Linux Mozilla");
$m->add_header( Cookie => 'BrowserDetect=passed' );
$m->get('http://editorial.tele2internet.fr/?page=ETAT_RESEAU&popup=1');
exit unless $m->success;

# sélection de la sortie
if (@ARGV) {
    open STDOUT, ">", $ARGV[0]
      or warn "Impossible de rediriger STDOUT sur $ARGV[0]: $!\n";
}
# affichage des informations
my $tree = HTML::TreeBuilder->new_from_content( $m->content );
print
  map { s/\n*\x95/\n\n*/g; $_ }
  map { $_->as_trimmed_text . "\n" }
  $tree->look_down( _tag => 'span', class => qr/apptext/ );
```

En pratique, à chaque fois que j'ai été déconnecté depuis que je fais tourner ce script, je n'ai jamais trouvé d'explication dans les messages d'état du réseau ! Ça doit être le modem qui chauffe...

## Surveiller une page web

J'utilise depuis peu le logiciel de téléphonie en ligne Skype. Si Skype est gratuit, certains services sont payants (comme SkypeOut, qui permet de téléphoner vers des téléphones classiques). En me connectant par hasard sur mon profil récemment, j'ai vu un message parlant de *SkypeOut Gift Day*. Il s'agit apparemment d'une offre promotionnelle permettant justement de gagner des « crédits » d'appel pour SkypeOut. Tous les détails sont sur la page **http://www.skype.com/campaigns/freeskypedays/**. Il suffit de se connecter sur sa page personnelle Skype le bon jour et de cliquer au bon endroit. J'aimerais bien gagner ces crédits, mais j'ai plus intéressant à faire que de me connecter sur ma page Skype tous les jours... La page personnelle de Skype (en anglais) contient le message suivant : *Today is not SkypeOut Gift Day, but perhaps it's coming up soon?* (Sur la version française du site : « Aujourd'hui n'est pas une Journée cadeau SkypeOut... mais il y en aura une très bientôt ! »). Je vais donc programmer un robot qui va vérifier une ou deux fois par jour la page en question et me prévenir le jour où ce message n'apparaît plus. Ce jour-là, je ferai l'effort de me connecter moi-même et de voir ce qu'il y a à faire pour gagner ce crédit de temps. Pour accéder à ma page personnelle, il me faut tout d'abord m'identifier. Un peu de recherche me dirige vers le lien `https://secure.skype.com/store/member/login.html` qui contient le formulaire de connexion. C'est notre premier formulaire d'identification depuis le début de cet article !




*Le formulaire de login de Skype*

Appelons mech-dump à la rescousse :

```
$ mech-dump https://secure.skype.com/store/member/login.html
POST https://secure.skype.com/store/member/dologin.html
  username=                     (text)
  password=                     (password)
  login=Sign me in              (submit)
```

Voilà un formulaire simple, comme je les aime. :-)

En quelques lignes, nous pouvons produire un script de connexion et de recherche du message.

```
    # formulaire de login
    $m->get('https://secure.skype.com/store/member/login.html');
    die $m->res->status_line unless $m->success;
    # remplissage et validation
    $m->set_fields(
        username => 'monlogin', # entrez vos identifiants de
connexion ici
        password => 'monpasse',
    );
    $m->click;
    die $m->res->status_line unless $m->success;
    # un cadeau ?
    print localtime() . " - Connecte-toi à ta page Skype !\n"
      if $m->content !~ /Today is not SkypeOut Gift Day/;
```

On ne devrait pas avoir de problèmes à cause de la langue, car la sélection de celle-ci se fait par un cookie `language`. Lors de mes tests, j'ai pu constater qu'au moins une fois l'identification a échoué et que la page contenait le message d'erreur suivant : *New account creation is temporarily disabled. Please try again after a few minutes.*

L'ajout de la ligne :

```
    # fait passer notre robot pour Mozilla, grâce à l'en-tête
User-Agent:
    $m->agent_alias('Linux Mozilla');
```

a eu l'air de satisfaire le serveur Skype. Peu de temps après une nouvelle tentative de connexion **sans** cette ligne a réussi. Ce n'est donc pas l'en-tête `User-Agent:` qui est en cause. (Ceci dit, il peut être parfois utile de se faire passer pour un navigateur « standard » pour déjouer certaines tentatives de détection du navigateur.) Mais alors, comment vérifier que l'identification est correcte ? On pourrait supposer que Skype utilise la réponse `HTTP standard 403 Forbidden` en cas de problème d'identification. Il est facile de vérifier que dans tous les cas (mot de passe valide ou non) la réponse est un magnifique `200 OK` qui ne nous apporte donc aucune information (du moins en ce qui concerne l'identification : en cas de problème réseau, on aura une erreur 500 par exemple, qui sera détectée par la ligne utilisant `$m->success()`). Nous pourrions analyser la page reçue en réponse pour trouver des informations permettant de distinguer le succès de l'échec de connexion. Il y a beaucoup plus simple. La ligne suivante permet d'afficher les cookies reçus par notre robot :

```
    print $m->cookie_jar->as_string;
```

Nous voyons les cookies au format interne de `HTTP::Cookies` (c'est également sous cette forme qu'ils sont sauvés par la méthode `save()`) :

```
    Set-Cookie3: loggedin=1; path="/"; domain=.skype.com; path_
spec; expires="2005-07-27 15:02:40Z"; version=0
    Set-Cookie3: username=monlogin; path="/"; domain=.skype.com;
path_spec; expires="2005-07-27 15:02:40Z"; version=0
    Set-Cookie3: skype_store2=70b6ea6caeff30ab42d7f60f05d20dfd;
path="/"; domain=secure.skype.com; path_spec; discard; version=0
```

Le nom du cookie `loggedin` suffit à trahir son rôle (je me demande s'il n'y a pas un problème de sécurité, d'ailleurs). On vérifie facilement que les cookies `username` et `loggedin` ne sont pas envoyés par le serveur en cas d'erreur lors de l'identification.

Nous allons réaliser l'analyse des cookies reçus à l'aide de la méthode `scan()` de `HTTP::Cookies` (la class de `libwww-perl` qui gère les cookies). Cette méthode permet d'appliquer une fonction de rappel à tous les cookies stockés dans un objet `cookie_jar` (boîte à gâteaux, en anglais). La fonction de rappel attend les 10 paramètres suivants (dans l'ordre) : `version`, `key`, `val`, `path`, `domain`, `port`, `path_spec`, `secure`, `expires`, `discard` et `hash`. (Ce dernier paramètre, `hash`, existe à des fins d'extensiblité et permet de recevoir des paramètres non standards du cookie).

La vérification de la réussite de notre identification se fait de la façon suivante :

```
# vérification que l'identification est réussie
{
  my $ok = 0;
  $m->cookie_jar->scan( sub { $ok++ if $_[1] eq 'loggedin' } );
  die "Echec d'identification : mauvais Pseudo Skype ou mot de passe ?"
     unless $ok;
}
```

En ajoutant ce bout de code juste avant la ligne `print localtime()`..., notre script saura se connecter à Skype et vérifier que l'identification a réussi. Nous aurons donc deux messages différents selon que le site Skype a refusé notre connexion (comme dans le cas *New account creation is temporarily disabled*...) ou que la page personnelle a changé.

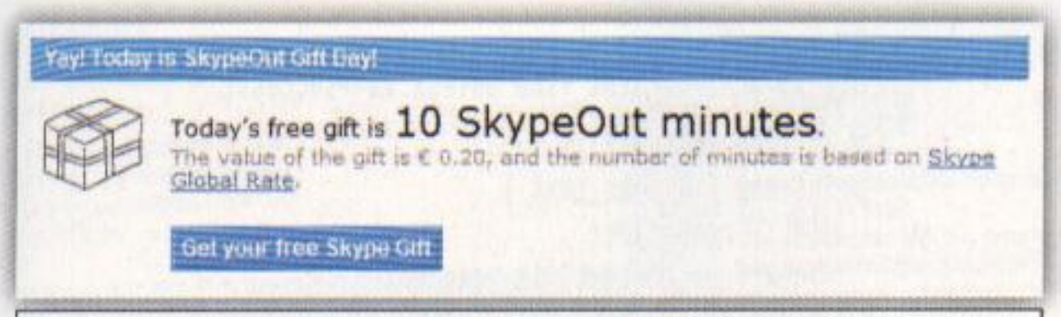
Après installation dans la crontab, il n'y a plus qu'à attendre d'être prévenu par un mail de *Cron Daemon* que mon cadeau Skype m'attend ! :-)

## À moi les cadeaux !

Quatre jours après avoir écrit les lignes ci-dessus, j'ai reçu un email laconique :

```
Sat Jul 30 15:25:12 2005 - Connecte-toi à ta page Skype !
```

Et voici ce qui m'attendait :




***L'annonce du cadeau de Skype***

Au point où nous en sommes, nous allons essayer d'automatiser aussi cette partie-là, histoire de ne plus rien avoir à faire du tout ! Cette partie ne nous laisse pas le droit à l'erreur, car on ne peut valider le formulaire qu'une seule fois.

Nous pouvons commencer par demander à notre script d'afficher tous les formulaires de la page avec :

```
print $_->dump for $m->forms;
```

La page ne contient qu'un seul formulaire :

```
POST https://secure.skype.com/store/myaccount/givepromotion.html
  promotion=4              (hidden readonly)
  <NONAME>=Get your free Skype Gift (submit)
```

C'est du tout cuit ! Notre script va donc regarder si la page contient un formulaire dont l'action est `givepromotion.html`. Si le formulaire existe, il va le valider et nous informer du

résultat ; dans le cas contraire, nous serons quand même informés qu'il y a quelque chose à faire. Le HTML de Skype est très propre et s'appuie énormément sur les CSS. En inspectant le HTML de la page contenant le formulaire, j'ai constaté que le formulaire était dans un bloc `<div class="freeskypeday">`. Avant de valider le formulaire, nous allons donc afficher le message d'information à l'aide de `HTML::TreeBuilder` :

```
print map { $_->as_text }
  HTML::TreeBuilder
    ->new_from_content( $m->content )
    ->look_down( _tag => 'div', class => 'freeskypeday' );
```

Comme l'objet `HTML::Tree` n'est utilisé qu'une fois pour afficher un message, on peut se passer de variable intermédiaire. Ensuite, si la page contient un formulaire qui correspond à celui du cadeau, on le valide :

```
if (   $m->current_form
    && $m->current_form->action =~ /givepromotion\.html$/ )
{
    $m->submit;
    die $m->res->status_line unless $m->success;
}
```

Une petite remarque : ne jamais oublier d'ajouter une ligne `print $m->content` quand on récupère une page pendant le développement d'un script. Ça permet de conserver une copie de la page reçue pour analyse. C'est ainsi qu'après validation, j'ai constaté qu'un message d'information était affiché entre balises `<h1>`. J'ai donc ajouté a posteriori la commande `print()` adéquate pour voir ce message la prochaine fois.

Il reste un dernier point à régler : le message change une fois qu'on a reçu le cadeau. On lit désormais : *Today is SkypeOut Gift Day, but it seems that you already received your gift* (sur la version française : « Aujourd'hui est un jour cadeau pour SkypeOut, mais il semble que vous ayez déjà reçu votre cadeau. »). Afin de ne pas être averti inutilement une fois que le cadeau a été retiré, nous allons également ignorer ce message.

En collant tous ces morceaux bout à bout et en rajoutant de quoi passer le login et le mot de passe sur la ligne de commande, nous obtenons le script suivant :

```
#!/usr/bin/perl
use strict;
use warnings;
use WWW::Mechanize;
use HTML::TreeBuilder;
use Getopt::Long;
no warnings 'utf8';
my %CONF;
GetOptions( \%CONF, "username=s", "password=s" ) or die << 'USAGE';
Options disponibles :
  --username
  --password
USAGE

my $m = WWW::Mechanize->new

# formulaire de login
$m->get('https://secure.skype.com/store/member/login.html');
die $m->res->status_line unless $m->success;

# remplissage et validation
$m->set_fields(
    username => $CONF{username},
    password => $CONF{password},
);
$m->click;
die $m->res->status_line unless $m->success

# vérification que l'identification est réussie
{
    my $ok = 0;
    $m->cookie_jar->scan( sub { $ok++ if $_[1] eq 'loggedin' } );
    die "Echec d'identification : mauvais Pseudo Skype ou mot de
passe ?\n"
      unless $ok;
}
# y a-t-il un cadeau pas encore récupéré ?
if (   $m->content !~ /Today is not SkypeOut Gift Day/
    && $m->content !~ /you already received your gift/ )
{

    # affiche le message
    print localtime() . " - Connecte-toi à ta page Skype !\n\n";
    print map { $_->as_text }
      HTML::TreeBuilder
      ->new_from_content( $m->content )
      ->look_down( _tag => 'div', class => 'freeskypeday' );

    # valide le formulaire
    if (   $m->current_form
        && $m->current_form->action =~ /givepromotion\.html$/ )
    {
        $m->submit;
        die $m->res->status_line unless $m->success;

        # affiche les détails
        print map { $_->as_text }
          HTML::TreeBuilder
          ->new_from_content( $m->content )
          ->look_down( _tag => 'h1' );
    }
}
```

Tout ce travail pour un cadeau de 20 centimes d'euro !

Épilogue : Un peu plus d'une semaine après que j'ai récupéré mon cadeau, la page principale du compte Skype ne contient plus aucune information au sujet des cadeaux SkypeOut.

Évidemment, mon script m'a prévenu que la page avait changé. J'ai ainsi pu constater qu'il était devenu probablement inutile... jusqu'à la prochaine campagne publicitaire de Skype.

*Merci aux Mongueurs de Perl et à Michel Grafmeyer pour leur relecture attentive. Le code des trois scripts présentés dans cet article, ainsi que celui du proxy espion sont présents sur le CD.*

**Philippe « BooK » Bruhat,**

**book@mongueurs.net**

# Le langage Ada – 5
## Les enregistrements

Yves Bailly

**EN DEUX MOTS** **Les tableaux du mois dernier ne permettent de stocker que des informations d'un même type. Comme beaucoup d'autres langages, Ada offre un moyen de placer en une seule entité des données de types différents. Là où le langage C parle de structures ou Python de classes, Ada a repris la terminologie du Pascal et parle plutôt d'enregistrements.**

L'enregistrement est la structure de donnée fondamentale en Ada pour représenter les idées du programmeur. Doté d'une grande souplesse, on imagine mal comment un programme non trivial pourrait s'en passer, à moins de tolérer une effroyable complexité dans l'écriture du code. Pour l'essentiel, ce que nous allons voir ici n'est pas propre à Ada. On retrouve nombre des aspects dans nombre de langages – mais quelques possibilités sont uniques à Ada.

## Déclaration

Supposons que nous souhaitions réaliser un programme pour jouer aux Tours de Hanoï (toute ressemblance avec d'autres articles n'aurait rien de fortuit). Il nous faut une structure pour représenter une tour, qui contiendra essentiellement un tableau de disques et le nombre de disques actuellement empilés sur la tour. Ce qui pourrait ressembler à ceci :

```
1 type Disque is new Integer range 0..10 ;
2 type Étage is new Integer range 0..10 ;
3 type Pile_Disques is array (Étage) of Disque ;
4
5 type Tour is
6 record
7    dernier_étage: Étage := 0 ;
8    pile: Pile_Disques := (others=>0) ;
9 end record ;
```

Les trois premières lignes définissent simplement quelques types qui nous seront utiles par la suite, la pile de disque étant représentée par un simple tableau (ligne 3). Le véritable objet de notre attention aujourd'hui se situe entre les lignes 5 et 9. Nous définissons ici un type nommé `Tour`, qui est un enregistrement (`record`) contenant deux champs nommés `dernier_étage` (de type `Étage`) et `pile` (de type `PileDisques`).

Petite remarque en passant : vous aurez peut-être remarqué les accents présents dans ce code source, ce qui peut paraître incongru. Je me suis permis cette fantaisie pour montrer une possibilité du compilateur GNAT, qui est justement d'accepter ce genre de caractères dans les identifiant.

En fait, cela fera bientôt partie de la norme Ada officielle : les compilateurs de la prochaine version du langage devraient être capables de traiter un code source contenant des noms de variables composés de caractères grecs et turcs (ce qui ne manquerait pas de piquant), ou autres, naturellement. Il est pour l'instant nécessaire de donner l'option `-gnatif` au compilateur, par exemple :

```
$ gnatmake -gnatif hanoi_decl.adb
```

Mais revenons à notre structure. Cela ressemble fort à une déclaration `struct` du langage C, avec toutefois cette différence notable qu'il est possible de donner une valeur par défaut pour les champs, ce qui est vivement recommandé.

On reproduit ainsi un comportement comparable à celui du constructeur d'une classe C++ ou Java, quoiqu'en plus limité.

Naturellement, rien ne nous empêche d'utiliser ce type pour composer d'autres types, par exemple :

```
1 type Num_Tour is new Integer range 1..3 ;
2 type Rangée_Tours is array (Num_Tour) of Tour ;
3 type Num_Mouvements is new Integer range 0..Integer'Last ;
4 type Jeu_Hanoï is
5 record
6    tours: Rangée_Tours ;
7    nb_mouvements: Num_Mouvements := 0 ;
8 end record ;
```

La ligne 2 déclare un type tableau `Rangée_Tours` dont les éléments sont de type `Tour`, ce tableau étant lui-même utilisé dans un enregistrement `Jeu_Hanoï` contenant différents paramètres du jeu. Remarquez que nous n'avons pas donné de valeur initiale pour le premier champ.

Non pas que cela ne serait pas possible (nous allons bientôt voir comment), simplement ce n'est pas strictement nécessaire car les éléments du tableau sont des enregistrements eux-mêmes déjà initialisés. On pourrait objecter à cela que le jeu risque de se retrouver ainsi dans un état incohérent, sans aucun disque sur aucune tour...

## Utilisation

La déclaration d'une variable d'un type enregistrement se fait comme n'importe quelle autre :

```
jeu: Jeu_Hanoï;
```

De même que pour les tableaux, on peut donner un agrégat pour initialiser les composants de l'enregistrement, sauf qu'au lieu de donner l'indice de l'élément on donne son nom.

Dans notre cas, nous avons un enregistrement qui contient un tableau d'enregistrements contenant un tableau. Une initialisation complète pourrait ressembler à ceci :

```
jeu: Jeu_Hanoï := (
   tours => (
      1 => (
         dernier_étage => 10,
         pile => (0, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1)),
      others => (
         dernier_étage => 0,
         pile => (others => 0))),
   nb_mouvements => 0) ;
```

L'accès aux éléments d'un enregistrement se fait en utilisant la notation pointée usuelle. Voici par exemple une procédure *qui* affiche le contenu d'une instance de **`Jeu_Hanoï`** :

```
 1 procedure Afficher(jeu: in Jeu_Hanoï) is
 2 begin
 3  Loop_Tours:
 4  for ind_tour in Num_Tour
 5  loop
 6    Put("Tour " & Num_Tour'Image(ind_tour) & " : ") ;
 7    Loop_Etages:
 8    for ind_etage in 1..Étage'Last
 9    loop
10      Put(Disque'Image(jeu.tours(ind_tour).pile(ind_etage))) ;
11    end loop Loop_Etages ;
12    New_Line ;
13  end loop Loop_Tours ;
14 end Afficher ;
```

Remarquez que rien dans cette portion de code ne suggère que nous ayons un maximum de 10 disques, selon la définition des types `Étage` et `Tour`. La ligne 10 montre l'accès aux différents éléments : `jeu.tours` est le tableau des tours, `jeu.tours(ind_tour)` désigne l'enregistrement de type `Tour` à l'indice `ind_tour` dans ce tableau, etc.

Les enregistrements supportent de plus les opérations élémentaires que sont l'affectation (`:=`) et la comparaison (`=` et `/=`). Ces opérations sont appliquées récursivement sur chacun des éléments de l'enregistrement. Par exemple, si la longue valeur d'initialisation de la déclaration plus haut vous rebute, voici une procédure qui permet d'initialiser une variable de type `Jeu_Hanoï`, en fonction du nombre initial de disques que l'on veut sur la première tour :

```
 1 procedure Init(jeu:     out   Jeu_Hanoï ;
 2                 nb_init: in    Étage) is
 3    jeu_tmp: Jeu_Hanoï ;
 4 begin
 5    jeu_tmp.tours(1).dernier_étage := nb_init ;
 6    for ind_etage in 1..nb_init
 7    loop
 8       jeu_tmp.tours(1).pile(ind_etage) :=
 9          Disque(Étage'Last-(ind_etage-1)) ;
10    end loop ;
11    jeu := jeu_tmp ;
12 end Init ;
```

Le passage par une variable temporaire nous assure que tous les éléments auront au moins une valeur par défaut. Le traitement se limite donc aux éléments à modifier selon le paramètre `nb_init`. Le résultat de l'initialisation est renvoyé dans le paramètre en sortie `jeu` grâce à l'affectation de la ligne 11.

Remarquez le transtypage ligne 9 : il est nécessaire, car le membre `pile` contient des éléments de type `Disque`, tandis que `ind_etage` et `Étage'Last` sont de type `Étage`. Et encore une fois, nous n'avons pas utilisé explicitement la limite de 10 du nombre de disques : elle est tout simplement récupérée par l'attribut `Last` du type `Étage`.

## Représentation

Une différence fondamentale entre des langages comme le C et Ada est que ce dernier n'impose rien quant à la représentation en mémoire des structures de données déclarées dans le programme. Le compilateur a toute latitude pour stocker tout cela en mémoire comme bon lui semble, en fonction des options qui lui sont données pour minimiser l'encombrement mémoire ou optimiser la vitesse d'exécution. Par exemple, un enregistrement ne contenant que deux entiers d'un type borné entre 1 et 10 pourrait parfaitement n'occuper qu'un seul octet, ce qui est binairement suffisant ; un tableau de six booléens (type `Boolean`) pourrait n'occuper que six bits, c'est-à-dire moins d'un octet ; au contraire, notre enregistrement de deux entiers pourrait s'étaler sur 16 octets, si on demande la meilleure performance possible et que le matériel sous-jacent est plus efficace pour accéder aux données tous les 8 octets... Ne soyez pas étonnés de cela, Ada a été conçu pour pouvoir être utilisé pour des logiciels embarqués (comme les sondes spatiales) où les contraintes et limites matérielles sont parfois très fortes.

*Une différence fondamentale entre des langages comme le C et Ada est que ce dernier n'impose rien quant à la représentation en mémoire des structures de données déclarées dans le programme.*

Mais parfois ces « aléas » dans la représentation mémoire ne sont pas souhaitables : on peut vouloir contrôler très précisément comment les données sont agencées en mémoire. Ada propose différents moyens pour cela, par le biais d'attributs et de directives `pragma`. Cela peut s'avérer fort utile si le programme doit manipuler directement un périphérique par ses ports d'entrées/sorties.

Voyons un petit programme d'exemple :

```
 1 with Text_IO ; use Text_IO ;
 2 procedure test_bits is
 3    type Petit is new Integer range 1..5 ;
 4 --   for Petit'Size use 5 ;
 5    type Enreg is
 6    record
 7       a: Petit ;
 8       b: Boolean ;
 9       c: Petit ;
10       d: Petit ;
11       e: Boolean ;
12    end record ;
13 --   pragma Pack(Enreg) ;
14    type Tab1 is array (Petit) of Petit ;
15 --   for Tab1'Component_Size use 10 ;
16    type Tab2 is array (Petit) of Enreg ;
```

```
17 --   pragma Pack(Tab2) ;
18 e: Enreg ;
19 begin
20    Put_Line("Petit'Size = " &
21       Integer'Image(Petit'Size)) ;
22    Put_Line("Enreg'Size = " &
23       Integer'Image(Enreg'Size)) ;
24    Put_Line("Enreg.a'First_Bit = " &
25       Integer'Image(e.a'First_Bit)) ;
26    Put_Line("Enreg.b'First_Bit = " &
27       Integer'Image(e.b'First_Bit)) ;
28    Put_Line("Enreg.c'First_Bit = " &
29       Integer'Image(e.c'First_Bit)) ;
30    Put_Line("Enreg.d'First_Bit = " &
31       Integer'Image(e.d'First_Bit)) ;
32    Put_Line("Enreg.e'First_Bit = " &
33       Integer'Image(e.e'First_Bit)) ;
34    Put_Line("Enreg.a'Position = " &
35       Integer'Image(e.a'Position)) ;
36    Put_Line("Enreg.b'Position = " &
37       Integer'Image(e.b'Position)) ;
38    Put_Line("Enreg.c'Position = " &
39       Integer'Image(e.c'Position)) ;
40    Put_Line("Enreg.d'Position = " &
41       Integer'Image(e.d'Position)) ;
42    Put_Line("Enreg.e'Position = " &
43       Integer'Image(e.e'Position)) ;
44    Put_Line("Tab1'Size = " &
45       Integer'Image(Tab1'Size)) ;
46    Put_Line("Tab1'Component_Size = " &
47       Integer'Image(Tab1'Component_Size)) ;
48    Put_Line("Tab2'Size = " & Integer'
49       Image(Tab2'Size)) ;
50    Put_Line("Tab2'Component_Size = " &
51       Integer'Image(Tab2'Component_Size)) ;
52 end test_bits ;
```

Les lignes commentées (4, 13, 15 et 17) contiennent des instructions précisant la représentation mémoire à adopter pour différents objets. Le programme affiche ensuite simplement des informations sur la taille et la position de ces objets. Aspect essentiel à bien garder en mémoire : les tailles et positions sont données en bits et non en octets.

Les attributs impliqués ici sont :

- `Size` donne la taille (en bits, donc – voir [AARM 13.3-40,45]) de l'objet lorsqu'il est interrogé (par exemple ligne 21) ou permet de la spécifier (par exemple ligne 4) ;
- `Component_Size` ne concerne que les types tableaux et permet d'obtenir (ligne 47) ou de définir (ligne 15) la taille occupée par chaque élément dans le tableau ; à noter qu'il est permis à l'implémentation [AARM 13.3-73] d'adapter la valeur donnée ;
- `First_Bit` [AARM 13.5.2] est appliqué à un élément d'un enregistrement et donne le décalage du premier bit de cet élément par rapport au début de l'octet mémoire qui le contient (en nombre de bits) ; l'attribut `Last_Bit` donne le décalage du dernier bit ; ces deux attributs ne peuvent pas être définis (enfin, pas directement) ;
- Enfin, `Position` donne la position d'un élément dans un enregistrement, cette fois exprimée en octets ; cet attribut ne peut pas être défini.

*Répétons-le : tout n'est que souplesse en Ada.*

Tout cela nous donne déjà pas mal de souplesse. Mais ce n'est pas tout. Le langage Ada fait un usage assez intensif des directives `pragma`, dont le rôle est essentiellement de donner des informations ou des conseils au compilateur. La première que nous rencontrons est la directive `pragma Pack` (lignes 13 et 17) : elle indique au compilateur qu'il doit tenter de minimiser l'encombrement mémoire des instances du type auquel elle est appliquée, qui doit être un type composé (tableau ou enregistrement). C'est une recommandation forte, qui doit être suivie même au prix de la vitesse d'exécution.

Pour fixer les idées, exécutez le programme précédent en jouant sur les lignes en commentaires.

La troisième version du programme, si elle devait créer beaucoup d'instances du type `Enreg` dans un tableau, occuperait beaucoup moins de mémoire (presque neuf fois moins) que la première version. Par contre, elle serait sensiblement plus lente, car il est nécessaire d'ajouter des instructions de calcul et manipulations binaires pour accéder aux éléments. Un test rapide montre que le parcours d'un tableau de 10000 éléments est presque quatre fois plus lent dans la première version (plus de huit fois si on compile avec l'optimisation -O2).

Il existe encore d'autres possibilités pour affiner la représentation mémoire : consultez la norme pour plus de détails.

## Erratum

Je profite de ce chapitre sur les représentations des données pour m'excuser d'une erreur qui s'est glissée dans l'article du moins dernier, au sujet des types énumérés, précisément ce membre de phrase : « il n'est pas possible d'associer explicitement une valeur numérique à une valeur symbolique ». C'est tout simplement faux : la section 13.4 du AARM expose justement une syntaxe permettant d'effectuer une telle association. Par exemple :

```
type TT is (AA, BB, CC) ;
for TT use (AA=>25, BB=>3213, CC=>6598) ;
```

Les symboles AA, BB et CC du type TT seront représentés en mémoire par les entiers 25, 3213 et 6598.

## Paramétrisation

Répétons-le : tout n'est que souplesse en Ada. Il est ainsi possible de paramétriser un enregistrement, selon un ou plusieurs critères. Le cas le plus simple consiste à utiliser une valeur pour dimensionner un tableau non contraint. Prenons un exemple :

```
1 procedure Record_Param is
2    type TTab is array (Integer range <>) of Integer ;
3    type TRec(taille: Integer) is
4    record
5       t: TTab(1..taille) := (others => 0) ;
6    end record ;
7 nb: Integer := 10 ;
8 r1: TRec(nb) ;
```

**Effets d'instructions de représentation**

| TOUTES LES LIGNES DÉSACTIVÉES | LIGNES 4 ET 17 ACTIVÉES | TOUTES LES LIGNES ACTIVÉES |
|---|---|---|
| Petit'Size = 3<br>Enreg'Size = 136<br>Enreg.a'First_Bit = 0<br>Enreg.b'First_Bit = 0<br>Enreg.c'First_Bit = 0<br>Enreg.d'First_Bit = 0<br>Enreg.e'First_Bit = 0<br>Enreg.a'Position = 0<br>Enreg.b'Position = 4<br>Enreg.c'Position = 8<br>Enreg.d'Position = 12<br>Enreg.e'Position = 16<br>Tab1'Size = 160<br>Tab1'Component_Size = 32<br>Tab2'Size = 800<br>Tab2'Component_Size = 160 | Petit'Size = 5<br>Enreg'Size = 40<br>Enreg.a'First_Bit = 0<br>Enreg.b'First_Bit = 0<br>Enreg.c'First_Bit = 0<br>Enreg.d'First_Bit = 0<br>Enreg.e'First_Bit = 0<br>Enreg.a'Position = 0<br>Enreg.b'Position = 1<br>Enreg.c'Position = 2<br>Enreg.d'Position = 3<br>Enreg.e'Position = 4<br>Tab1'Size = 40<br>Tab1'Component_Size = 8<br>Tab2'Size = 200<br>Tab2'Component_Size = 40 | Petit'Size = 5<br>Enreg'Size = 17<br>Enreg.a'First_Bit = 0<br>Enreg.b'First_Bit = 5<br>Enreg.c'First_Bit = 6<br>Enreg.d'First_Bit = 3<br>Enreg.e'First_Bit = 0<br>Enreg.a'Position = 0<br>Enreg.b'Position = 0<br>Enreg.c'Position = 0<br>Enreg.d'Position = 1<br>Enreg.e'Position = 2<br>Tab1'Size = 50<br>Tab1'Component_Size = 10<br>Tab2'Size = 88<br>Tab2'Component_Size = 17 |
| Remarquez comme la taille de Petit est petite, seulement 3 bits : le compilateur a déterminé que c'était suffisant pour contenir les valeurs de ce type.<br>Dans l'enregistrement, tous les First_Bit sont à zéro : les composants commencent tous à une limite d'octet.<br>La taille des éléments du type Tab1 nous montre qu'en réalité Petit occupe 4 octets en mémoire. | La ligne 4 impose une taille pour Petit : celle-ci influe naturellement sur le reste.<br>En particulier, l'enregistrement est plus petit : le fait de donner une taille à Petit réduit son encombrement, bien que cette taille soit plus grande que la taille minimum déterminée précédemment.<br>Mais il y a encore beaucoup de vide dans tout cela... | Cette fois l'encombrement de l'enregistrement est véritablement minimisé : il n'occupe plus que 17 bits, soit à peine plus que deux octets. Il y a encore un peu de perte dans Tab2 (3 bits), parce que le compilateur refuse de créer des tableaux d'une taille non multiple de 8 étant donné le matériel sous-jacent (en l'occurrence, un Intel Pentium 4). L'augmentation de la taille de Tab1 vient du fait qu'on a imposé une taille « trop » grande à ses éléments (ligne 15). |

```
 9 r2: TRec(nb) ;
10 r3: TRec(2*nb) ;
11 begin
12    r1 := r2 ;
13    r1 := r3 ;
14 end Record_Param ;
```

Nous déclarons ligne 2 un type tableau non contraint TRec. Il est alors normalement interdit d'utiliser ce type comme élément d'un autre tableau ou d'un autre enregistrement, à moins de le contraindre. Les lignes 3 à 6 définissent un type enregistrement TRec, qui contient justement un membre de ce type tableau. Celui-ci est contraint grâce à un paramètre donné à TRec, nommé taille. Dans le cadre d'un enregistrement, on appelle un tel paramètre un discriminant. Celui-ci fait partie intégrante de TRec, sa valeur étant obtenue lors de l'instanciation du type (lignes 8 à 10). Il joue alors le rôle d'un membre, que l'on peut consulter comme n'importe quel autre (par exemple avec r1.taille), mais dont on ne peut modifier la valeur : une instruction comme r1.taille := 5 est interdite. Remarquez que la valeur du discriminant n'est pas nécessairement statique, elle peut être issue d'une variable ou d'un calcul.

Les affectations entre instances des lignes 12 et 13 sont syntaxiquement valides. Toutefois, si la première ne pose aucun problème, la seconde provoquera la levée d'une exception (nommément Constraint_Error) : les affectations entre enregistrements contenant des discriminants ne sont en effet possibles que si les valeurs des discriminants sont égales. Les discriminants peuvent également être utilisés pour adapter l'apparence d'un type enregistrement. Imaginons un instant que nous voulions réaliser une petite application pour stocker notre vaste collection de livres et de films. Ces deux types d'information ont au moins une chose en commun : le titre. Par contre, on peut vouloir stocker le nombre de pages pour un livre, mais la durée pour un film. Il nous faudrait alors définir deux types différents pour les représenter... à moins d'utiliser un discriminant :

```
 1 type Type_Info is (Livre, Film) ;
 2 type Info(nature: Type_Info) is
 3 record
 4    titre: String(1..200) := (others=>' ') ;
 5    case nature is
 6       when Livre =>
 7          nb_pages: Integer ;
 8          isbn: String(1..13) ;
 9       when Film =>
10          durée: Float ;
11    end case ;
12 end record ;
```

La syntaxe utilisée est très similaire à celle du choix multiple. Selon la valeur du discriminant nature, l'enregistrement Info présentera

différents visages. Si cette valeur est le symbole `Livre`, alors `Info` contiendra (en plus du titre) les éléments `nb_pages` et `isbn`. Si cette valeur est `Film`, alors `Info` contiendra `durée`. Ce procédé est à rapprocher des types `union` du C/C++. Remarquez que les différents visages d'un enregistrement ainsi « discriminé » ne doivent pas nécessairement contenir des types compatibles ou un même nombre de champs.

L'utilisation d'un tel type se fait ainsi :

```
1 info1: Info(Livre) ;
2 info2: Info(Film) ;
3 -- .....
4 info1.nb_pages := 200 ;
5 info2.durée := 1.5 ;
6 info2.nb_pages := 1 ;
```

La valeur du discrimant est donnée à la déclaration, puis on accède normalement aux éléments. Remarquez la dernière ligne : elle est syntaxiquement correcte, mais comme `info2` a été déclaré comme étant un `Film`, il ne montre pas de champs nommé `nb_pages`. À l'exécution, l'exception `Contraint_Error` sera levée. Il est également possible de donner une valeur initiale lors de la déclaration d'une variable, ce qui est en fait le seul cas où l'affectation au discriminant est autorisée :

```
1 info3: Info := (
2    nature => Livre,
3    titre => "Titre" & Chaine_Vide(1..195),
4    nb_pages => 100,
5    isbn => "1-2345-6789-A") ;
```

La valeur du discriminant est donnée dans la liste d'initialisation (bien qu'il n'eût pas été incorrect de la répéter dans le nom du type en ligne 1). Cette fois, si on donne une valeur à un élément qui n'existe pas selon le discriminant donné, on obtient une erreur de compilation. De même, si la valeur du discriminant n'est pas donnée avec le nom du type, la liste d'initialisation est obligatoire. Il est ainsi interdit de déclarer une variable simplement avec `info4: Info;`, sauf dans le cas d'un paramètre de sous-programme, par exemple :

```
1 procedure Afficher(i: Info) is
2 begin
3    Put_Line("Titre = " & i.titre) ;
4    case i.nature is
5       when Livre =>
6          Put_Line("ISBN = " & i.isbn) ;
7       when Film =>
8          Put_Line("Durée = " & Float'Image(i.durée)) ;
9    end case ;
10 end Afficher ;
```

Dernier mot, rien n'interdit de déclarer plusieurs discriminants de types différents pour un type enregistrement, de donner une valeur par défaut ou d'utiliser le mécanisme des paramètres nommés lors de la déclaration d'une variable. La syntaxe des discriminants d'enregistrement est en fait assez proche de celle des déclarations et appels de sous-programmes.

## Extension

Dernier aspect des enregistrements que nous verrons aujourd'hui, en manière d'anticipation sur de prochains articles : les enregistrements marqués (`tagged`) et l'extension de leur contenu. Prenons un exemple classique, un enregistrement décrivant un cercle :

```
type Cercle is tagged record
   x: Float ;
   y: Float ;
   rayon: Float ;
end record ;
```

Un enregistrement classique, mais remarquez tout de même la présence du mot `tagged` dans la définition. Maintenant nous voudrions une ellipse, décrite par un centre, un petit rayon et un grand rayon. Il serait naturellement possible de recréer un type pour cela. Mais il serait peut-être plus intéressant de conserver le lien de parenté qui existe entre un cercle et une ellipse... c'est-à-dire, de construire notre ellipse comme une extension de notre cercle. Comme ceci :

```
type Ellipse is new Cercle with
record
   grand_rayon: Float ;
end record ;
```

Ceci définit un type `Ellipse` comme étant un nouveau `Cercle` contenant en plus un élément nommé `grand_rayon`. Selon cette représentation, une `Ellipse` « est aussi » un `Cercle` (ce qui est discutable sur le plan mathématique, mais c'est une autre histoire). Dit autrement, le type `Cercle` est l'ancêtre du type `Ellipse` ou encore le type `Ellipse` dérive du type `Cercle`... Certains auront probablement reconnu là un vocabulaire habituellement utilisé pour parler de programmation objet. Bonne nouvelle, c'est précisément de cela qu'il s'agit : ce mécanisme, introduit par la norme Ada95 (donc absent de la norme Ada83), est destiné à apporter à Ada les fonctionnalités d'un langage orienté objet. Ce que nous venons de voir n'est rien de plus que de l'héritage. Ada95 ne supporte que l'héritage simple. La prochaine norme Ada apportera la notion d'« interfaces », inspirée par le langage Java, réalisant ainsi une forme limitée d'héritage multiple. Rassurez-vous, les aspects objet de Ada feront l'objet d'un article spécifique, voire de plusieurs.

*La prochaine norme Ada apportera la notion d'« interfaces », inspirée par le langage Java, réalisant ainsi une forme limitée d'héritage multiple.*

## Conclusion

Voilà en ce qui concerne les enregistrements en Ada. Comme le laisse supposer la dernière section, nous les retrouverons bientôt – en fait, nous les utiliserons sans cesse au fil de nos découvertes. La prochaine fois, nous aborderons le mécanisme des paquetages, par lequel Ada permet les compilations séparées et la définition et la diffusion de composants réutilisables – autrement dit, nous découvrirons les bibliothèques en Ada.

**Yves Bailly,**

***http://wwwikafka-fr.net***

# Réaliser un client mail en Common Lisp

Philippe Brochard

**EN DEUX MOTS** **Dans cet article, je vous propose de réaliser un client mail en Common Lisp [1]. Celui-ci n'a pas la prétention de remplacer les clients mail existants. En effet, ce client ne fait que télécharger les sujets des mails et quelques lignes (pour aller vite avec un modem 56k par exemple) et propose de détruire les spams sur le serveur en laissant les mails intéressants intacts que l'on pourra rapatrier ensuite avec un vrai client mail. Nous allons donc voir une manière de dialoguer avec un serveur de mails (POP3), comment repérer les spams grâce à un filtre bayésien et enfin comment faire une interface graphique pour confirmer la suppression des spams.**

## Quelques outils pour la suite

Tout d'abord, avant de rentrer dans le vif du sujet, nous allons avoir besoin d'un certain nombre d'outils afin de manipuler les mails. Nous allons donc créer un premier *package* (`util.lisp`) contenant ces outils. Ce package contiendra aussi la définition des variables globales utilisées dans toute la suite du programme. Nous verrons, dans un autre paragraphe, que ce package contiendra aussi les fonctions permettant de sauvegarder les différentes variables du programme.

```
;;; --- util.lisp --- ;;;
(in-package :common-lisp-user)

(defpackage :util
  (:use :common-lisp)
  (:export :*mailbox* :*spam* :*ham* :*wrong-word-list*
     :*default-body-length*
     :make-message
     :message-subject :message-body
     :message-from :message-to
     :message-spam-proba
     :message-frame :message-bspam :message-bsubject
     :first-word-p :split-string
     :default-mailbox :make-mailbox
     :mailbox-host :mailbox-user :mailbox-password
     :mailbox-string
     :save-base :load-base
     :save-lisp-image))

(in-package :util)

(defstruct mailbox host user password)

(defstruct message to from subject body spam-proba
frame bspam bsubject)

(defparameter *mailbox*
  (list (make-mailbox :host "pop.mon_fai.fr"
                      :user "nom_utilisateur"
                      :password "mot_de_passe")
        (make-mailbox :host "pop.azerty.net"
                      :user "plop"
                      :password "pss")))

(defparameter *default-body-length* 10)

(defparameter *wrong-word-list* (list "vlagra" "bite"
"sexe" "sex"))
(defparameter *spam* (make-hash-table :test #'equalp))
(defparameter *ham* (make-hash-table :test #'equalp))

(defparameter *default-rcfile*
  (merge-pathnames (make-pathname :name ".clmailrc" :type "lisp")
                   (user-homedir-pathname)))

(defun first-word-p (word string)
  (let ((pos (search word string)))
    (and (numberp pos) (zerop pos))))
(defun split-string (string &optional (separator #\Space))
  (loop for i = 0 then (1+ j)
        as j = (position separator string :start i)
        as sub = (subseq string i j)
        unless (string= sub "") collect sub
        while j))

(defun default-mailbox ()
  (first *mailbox*))

(defun mailbox-string (&optional (mailbox (default-mailbox)))
  (format nil "Boite: ~A - Utilisateur: ~A"
          (mailbox-host mailbox) (mailbox-user mailbox)))
```

`*mailbox*` est une liste contenant les informations sur les différentes boites aux lettres. Chaque boite aux lettres est définie comme une structure contenant l'adresse du serveur hôte (`"pop.mon_fai.fr"`), le nom de l'utilisateur (son identifiant) et le mot de passe de l'utilisateur. Attention, ici le mot de passe est stocké en clair, il sera donc sûrement judicieux de fixer les permissions sur le fichier `util.lisp` afin d'en limiter l'accès, même si le mot de passe circulera en clair sur le réseau avec le protocole POP3. `*default-body-length*` correspond au nombre de lignes du mail que l'on veut télécharger en plus de l'en-tête. Vient ensuite la définition des variables globales concernant le filtre antispam. `*wrong-word-list*` contient les mots indésirables qui indiquent que le mail est sûrement un spam (comme par exemple `"viagra"`). `*spam*` est une table de *hashage* contenant les mots trouvés dans les spams et `*ham*` est une table contenant les mots trouvés dans les *hams* (les mails qui ne sont pas des spams).

Nous définissons aussi la variable `*default-rcfile*` qui contiendra l'emplacement du fichier de configuration `.clmailrc.lisp`. Pour cela, nous utilisons la fonction `user-homedir-pathname` qui retourne le répertoire `"maison"` de l'utilisateur courant. Par exemple avec GNU/Linux, le répertoire est `#P"/home/phil/"`, sous Windows `#P"C:\\Documents and settings\\phil\\"` et avec MacOS `#P"/Users/phil/"`. Et nous créons l'emplacement du fichier de configuration en associant le répertoire maison au fichier `".clmailrc.lisp"`.

La structure associée à un message est définie grâce à la fonction `defstruct`. Un message est constitué d'une destination (`to`), d'une origine (`from`), d'un sujet (`subject`), d'un corps (`body`), d'une probabilité d'être un spam (`spam-proba`) et de quelques éléments pour l'interface graphique (`frame`, `bspam`, `lsubject`).

Nous définissons l'inévitable fonction `split-string` qui permet de convertir une chaîne de caractères en une liste.

```
CL-USER> (split-string "bonjour ceci est une   chaîne de  caractère  !")
("bonjour" "ceci" "est" "une" "chaîne" "de" "caractère" "!")
```

Une remarque, nous aurions pu utiliser les expressions régulières pour faire une recherche plus fine sur les chaînes de caractères grâce au module `cl-ppcre` [2] (*portable Perl-compatible regular expressions for Common Lisp*) d'Edi Weitz, mais dans notre cas, la fonction `split-string` suffit.

Enfin, nous définissons quelques fonctions pour accéder plus facilement aux boites mail ou pour les afficher simplement. Par convention, la boite mail par défaut est la première de la liste `*mailbox*`.

## Dialoguer avec le serveur – Le protocole POP3

Maintenant que nous avons les variables globales et les petits outils, nous pouvons commencer à dialoguer avec le serveur POP3. Pour cela, nous définissons le module `pop3.lisp`.

```
;;; --- pop3.lisp --- ;;;
(in-package :common-lisp-user)

(defpackage :pop3
  (:use :common-lisp :util)
  (:export :connect-to-server
           :quit-server
           :stat-server
           :peek-mail
           :delete-mail))

(in-package :pop3)

(defun protect-line (line)
  (with-output-to-string (str)
    (loop for c across line do
          (princ (case c
                   (#\Escape "*")
                   (#\Return "")
                   (#\{ "\\{")
                   (#\} "\\}")
                   (t c)) str))
    str))

(defun read-line-from-server (sock &optional show check)
  (let ((line (protect-line (ignore-errors (read-line sock nil :
eof)))))
    (when show
      (format t "[Serveur]: ~S~%" line))
    (if check
        (when (first-word-p "+OK" line)
          line)
        line)))

(defun send-line-to-server (sock show control-string &rest args)
  (apply #'format show control-string args)
  (ignore-errors
    (apply #'format sock control-string args)
    (force-output sock)))

(defun connect-to-server (host user password &optional show)
  (let ((sock (ignore-errors (port:open-socket host 110))))
    (when sock
      (unless (read-line-from-server sock show t)
        (return-from connect-to-server :error-sock))
      (send-line-to-server sock show "USER ~A~%" user)
      (unless (read-line-from-server sock show t)
        (return-from connect-to-server :error-user))
      (send-line-to-server sock show "PASS ~A~%" password)
      (unless (read-line-from-server sock show t)
        (return-from connect-to-server :error-password)))
    sock))

(defun quit-server (sock &optional show)
    (send-line-to-server sock show "QUIT~%")
    (read-line-from-server sock show)
    (ignore-errors (close sock)))

(defun stat-server (sock &optional show)
  (send-line-to-server sock show "STAT~%")
  (let ((line (split-string (read-line-from-server sock show))))
    (values (or (parse-integer (second line) :junk-allowed t) 0)
            (or (parse-integer (third line) :junk-allowed t) 0))))

(defun end-of-line (line &optional (start 0))
  (string-trim " " (subseq line start)))

(defun peek-mail (sock num &optional show (N *default-body-
length*))
  (send-line-to-server sock show "TOP ~A ~A~%" num N)
  (let ((mesg (make-message :subject "(Sans titre)" :body "" :to
"Personne"
                            :from "Personne" :spam-proba 0)))
    (loop for line = (read-line-from-server sock show)
          with header = T
          until (or (eql line :eof) (string-equal line ".")) do
          (if header
              (cond
                ((first-word-p "To:" line)
                 (setf (message-to mesg) (end-of-line line 3)))
                ((first-word-p "From:" line)
                 (setf (message-from mesg) (end-of-line line 5)))
                ((first-word-p "Subject:" line)
                 (setf (message-subject mesg) (end-of-line line
8)))
                ((string= "" line) (setf header nil)))
              (setf (message-body mesg)
                    (format nil "~A~A~%" (message-body mesg)
line))))
    mesg))
(defun delete-mail (sock num &optional show)
  (send-line-to-server sock show "DELE ~A~%" num show)
  (first-word-p "+OK" (read-line-from-server sock show)))
```

Le dialogue entre un client et le serveur POP3 se fait au format texte, le serveur répondant aux commandes définies dans la RFC 1939.

**C'est-à-dire :**

- `USER nom` : identification de l'utilisateur ;
- `PASS mot_de_passe` : mot de passe de l'utilisateur ;
- `STAT` : obtient des informations sur la boite mail (nombre de mails et taille totale) ;
- `RETR N` : récupère le mail N ;
- `DELE N` : détruit le mail N sur le serveur ;
- `TOP N L` : récupère l'en-tête et les L lignes du mail N ;
- `QUIT` : ferme la connexion avec le serveur. Celui-ci procède alors à la suppression des mails marqués avec la commande `DELE` comme étant à supprimer.

Tout d'abord, nous devons nous connecter au serveur, ceci est réalisé grâce à la fonction `connect-to-server` qui crée une nouvelle *socket* (grâce au module `net.lisp` vu dans l'article précédent) et qui envoie le nom de l'utilisateur et le mot de passe au serveur grâce aux fonctions `read-line-from-server` et `send-line-to-server`. Ici, nous faisons un test d'erreur au cas où l'identification aurait échouée. En effet, dans le cas où tout s'est bien déroulé, le serveur envoie une ligne commençant par `"+OK"` et dans le cas contraire, il envoie une ligne commençant par `"-ERR"`.

La fonction `quit-server` permet de quitter le serveur et de fermer la socket proprement. Ce n'est qu'après avoir fermé le serveur proprement que la suppression des mails est effective.

Vient ensuite la fonction `stat-server` qui renvoie le nombre de mails et la taille totale des mails sur le serveur.

La fonction `peek-mail` est chargée de télécharger les informations importantes du mail comme le sujet, l'expéditeur et le destinataire. Pour cela, on utilise la commande `TOP` en ne demandant qu'à lire l'en-tête du mail et on remplit les champs de la structure message correspondants. Le serveur indique qu'il a fini d'envoyer l'en-tête par une ligne vide et qu'il a fini d'envoyer le message par une ligne ne contenant qu'un point. Si vous voulez tester d'autres champs de l'en-tête (comme par exemple le `Sender`), il vous suffit de rajouter une condition dans la fonction `cond`. De la même manière, si vous voulez appliquer le filtre antispam sur les N premières lignes du sujet, il suffit de changer le nombre de lignes à lire dans la commande `TOP` et adapter le filtre que nous allons voir par la suite pour qu'il tienne compte de ces nouvelles lignes.

Enfin, la fonction `delete-mail` permet de supprimer le mail N. Ici, il est impératif de savoir si la suppression a fonctionné ou non. En effet, le numéro du mail à supprimer correspond à la place du mail dans la liste `all-mails` que nous verrons par la suite. Donc, si la suppression a échoué, il faut en tenir compte pour garder la synchronisation entre la liste des mails et les mails présents sur le serveur. Pour cela, nous testons si le serveur renvoie une ligne commençant par `"+OK"` après la suppression du mail (nous aurions aussi pu tester si la ligne renvoyée était bien de type chaîne de caractères après avoir rajouté le test directement dans la fonction `read-line-from-server` avec l'argument optionnel `"check"`).

Après avoir défini ce module, nous pouvons le tester dans la REPL du Lisp :

```
CL-USER> (load "util.lisp")
CL-USER> (load "net.lisp")
CL-USER> (load "pop3.lisp")
CL-USER> (in-package :pop3)
POP3> (defparameter sock (connect-to-server
"pop.free.fr" "hocwp" "vous_ne
croyez_pas_que_j'allais_mettre_mon_mot_de_
passe_en_clair :)"))
SOCK
POP3> (stat-server sock t)
STAT
[Serveur]: +OK 3 2360
3
2360
POP3> (peek-mail sock 3)
#S(UTIL::MESSAGE :TO "hocwp@free.fr" :FROM "phil "
   :SUBJECT "where can find S_fsys_startup?"
   :BODY "slkdjskqdjskqldjsqklbcxnwcbnxwcbxnw,cbxw,nbcxw,n"
   :SPAM-PROBA 0 :FRAME NIL
   :BSPAM NIL :LSUBJECT NIL)
POP3> (peek-mail sock 3 t)
TOP 3 0
[Serveur]: +OK 792 octets
[Serveur]: Return-Path:
[Serveur]: Delivered-To: online.fr-hocwp@free.fr
[Serveur]: Received: (qmail 18546 invoked from network); 10 Jun
2005 09:21:13 -0000
[Serveur]: Received: from smtp-105-friday.nerim.net (HELO kraid.
nerim.net) (62.4.16.105)
[Serveur]:   by mrelay5-1.free.fr with SMTP; 10 Jun 2005 09:21:13
-0000
[Serveur]: Received: from grigri.elcforest (hocwp.net1.nerim.net
[213.41.153.61])
[Serveur]:         by kraid.nerim.net (Postfix) with ESMTP id
BF80540F75
[Serveur]:         for ; Fri, 10 Jun 2005 11:20:41 +0200 (CEST)
[Serveur]: Received: from phil by grigri.elcforest with local
(Exim 4.34)
[Serveur]:         id 1Dgfgv-00013U-HN
[Serveur]:         for hocwp@free.fr; Fri, 10 Jun 2005 11:20:41
+0200
[Serveur]: To: hocwp@free.fr
[Serveur]: Subject: where can find S_fsys_startup?
[Serveur]: Message-Id:
[Serveur]: From: phil
[Serveur]: Date: Fri, 10 Jun 2005 11:20:41 +0200
[Serveur]:
[Serveur]:
[Serveur]: "slkdjskqdjskqldjsqklbcxnwcbnxwcbxnw,cbxw,nbcxw,n"
[Serveur]: .
#S(UTIL::MESSAGE :TO "hocwp@free.fr" :FROM "phil "
   :SUBJECT "where can find S_fsys_startup?"
   :BODY "slkdjskqdjskqldjsqklbcxnwcbnxwcbxnw,cbxw,nbcxw,n"
   :SPAM-PROBA 0 :FRAME NIL
   :BSPAM NIL :LSUBJECT NIL)
POP3> (delete-mail sock 3)
T
POP3> (stat-server sock)
3
1568
POP3> (quit-server sock)
T
POP3> (defparameter sock (connect-to-server "pop.free.fr" "hocwp"
"..."))
SOCK
POP3> (stat-server sock)
2
1568
POP3> (quit-server sock)
T
POP3> (defparameter sock (connect-to-server "pop.free.fr" "hocwp"
"mauvais mot de passe"))
SOCK
POP3> sock
:ERROR-PASSWORD
POP3> (defparameter sock (connect-to-server "pouf" "hocwp" "pof"))
SOCK
POP3> sock
NIL
POP3>
```

Ici, nous voyons l'usage du paramètre optionnel "show" qui permet d'afficher ce que retourne le serveur dans les fonctions stat-server et peek-mail, ainsi que le code retourné par la fonction connect-to-server lorsque le mot de passe ou l'adresse du serveur POP3 sont incorrects. Sous Slime, si vous avez besoin de changer une fonction à la volée, il suffit de placer le curseur dans le corps de la fonction du fichier source (ici pop3.lisp).Vous pouvez soit l'évaluer avec la combinaison de touche [C-M-x] ([Contrôle] + [Alt] + [x]), soit la compiler avec [C-c C-c] ([Contrôle C] deux fois). [C-h m] ([Contrôle] [h] puis [m]) donnant (beaucoup) plus d'informations sur les combinaisons de touches du mode courant.

## Les filtres bayesiens : éliminer les mails indésirables

Maintenant que nous savons lire les mails sur un serveur POP3, il nous reste à voir comment éliminer ces @#{?§*£$ de spams ou pourriels. Pour cela, nous allons voir une première méthode naïve mais très efficace fondée sur une liste de mots indésirables et une deuxième méthode plus évoluée basée sur les filtres bayésiens. Pour cela, nous définissons le paquetage spam.lisp.

```
;;; --- spam.lisp --- ;;;
(in-package :common-lisp-user)

(defpackage :spam
  (:use :common-lisp :util)
  (:export :spam-proba
           :is-spam
           :learn-bad-word
           :learn-from-line
           :learn-from-file
           :spam :ham))

(in-package :spam)

(defun wrong-char-p (char)
  (not (or (standard-char-p char)
           (member char '(#\é #\è #\ç #\à #\ù #\â #\ô #\ê
                          #\î #\û #\ö #\ï #\ü #\ë #\ã)))))

(defun count-wrong-char (str)
  (loop for c across str
        count (wrong-char-p c)))

(defun find-wrong-word (str)
  (dolist (word *wrong-word-list*)
    (when (search word str :test #'string-equal)
      (return-from find-wrong-word t))))

(defun analyse-string (str)
  (or (> (count-wrong-char str) 3)
      (find-wrong-word str)))

(defun learn-bad-word (word)
  (push word *wrong-word-list*))

;;; Bayesian filter
(defun inc-word (word type)
  (let ((base (if (eq type :spam) *spam* *ham*)))
    (if (numberp (gethash word base))
        (incf (gethash word base))
        (setf (gethash word base) 1))))

(defun spam (word count)
  (setf (gethash word *spam*) count))

(defun ham (word count)
  (setf (gethash word *ham*) count))

(defun learn-from-line (line type)
  (unless (analyse-string line)
    (dolist (word (split-string line))
      (inc-word word type))))

(defun learn-from-file (filename type)
  (with-open-file (stream filename :direction :input)
    (loop for line = (read-line stream nil nil)
          while line do
          (learn-from-line line type))))

(defun word-proba (word)
  (let ((nspam (or (gethash word *spam*) 0))
        (nham (or (gethash word *ham*) 0)))
    (/ (+ 0.5 nspam) (+ 1 nspam nham))))

(defun analyse-spam (str)
  (loop for word in (split-string str)
        as N = 1 then (1+ N)
        as prob = (word-proba word)
        as p = (- 1 prob) then (* p (- 1 prob))
        as q = prob then (* q prob)
        finally (return (if (null N)
                            0.5
                            (let ((P1 (expt p (/ 1 N)))
                                  (Q1 (expt q (/ 1 N))))
                              (/ (- 1 P1) (- 2 P1 Q1)))))))

(defun spam-proba (str)
  (if (analyse-string str)
      1
      (analyse-spam str)))

(defun is-spam (proba)
  (when (numberp proba)
    (> proba 0.6)))
```

Tout d'abord, nous faisons en sorte de détecter les mots indésirables. Pour cela, nous définissons la fonction wrong-char-p qui permet de repérer des caractères indésirables dans une chaîne de caractères (ceux-ci provenant souvent de mails asiatiques). Nous définissons la fonction count-wrong-char qui permet de les compter. Et enfin, nous définissons la fonction find-wrong-word qui permet de repérer un mot indésirable dans une chaîne de caractères (ceux-ci sont stockés dans la liste *wrong-word-list*). De cette façon, grâce à la fonction analyse-string, nous détectons un spam lorsqu'il contient un mot indésirable ou trop de caractères illisibles. De plus, nous laissons la possibilité à l'utilisateur de rajouter des mots indésirables dans la liste grâce à la fonction learn-bad-word. Ceci est une méthode simpliste, mais qui élimine à coup sûr un nombre non négligeable de mails.

La deuxième méthode, plus évoluée, est basée sur les filtres bayésiens. Ceux-ci ont été mis en avant suite à l'article « *A plan for spam* » [3] de Paul Graham. Même si celui-ci n'est pas l'inventeur des filtres bayésiens et que l'algorithme utilisé dans son article n'est pas formellement correct, l'ensemble donne des résultats acceptables. De plus, ils sont utilisés dans des logiciels tel que Spamassassin [4] ou bogofilter [5]. Pour plus de détails, je vous laisse vous reporter au MISC N°8 et à l'article de Fabrice Rossi « Filtrage de SPAM par méthodes probabilistes » ainsi qu'à l'article du GNU/Linux Magazine N°60, « Découvrez les réseaux Bayésiens », de Jean-Michel Marin et Fabrice Rossi. Pour cette partie, je me suis inspiré de l'article « *Spam detection* » de Gary Robinson [6].

Pour faire un filtre bayésien, nous avons besoins de compter le nombre de fois où un mot apparaît dans un spam ou dans un ham (l'inverse d'un spam, soit un mail que l'on veut lire) afin de pouvoir calculer les probabilités de spam de ce mot. Pour cela, nous définissons la fonction `inc-word` qui permet d'augmenter le nombre de fois où un mot est apparu dans un spam ou un ham suivant le type du mot (`:spam` ou `:ham`). Nous définissons aussi les fonctions `spam` et `ham` qui permettent de fixer directement la fréquence d'apparition d'un mot dans un spam ou un ham.

Nous définissons ensuite la fonction `learn-from-line` qui permet d'enrichir la base des mots dans un spam ou un ham à partir d'une chaîne de caractères. Ainsi que la fonction `learn-from-file` qui permet de faire de même à partir d'un fichier contenant des phrases provenant de spams ou de hams. Vous pouvez trouver ce genre de base de spams sur spamarchive [7] et récupérer les sujets de ces spams après une petite moulinette à base de *sed* ou de macros d'emacs.

Maintenant que nous savons quel est le nombre de fois où un mot apparaît dans un spam ou dans un ham, nous pouvons calculer la probabilité que ce mot soit dans un spam. Pour cela, nous calculons le rapport :

$$p(mot = spam) = p(m) = \frac{N_{spam}}{N_{spam} + N_{ham}}$$

ou $N_{spam}$ et $N_{ham}$ sont le nombre de fois où le mot est apparu respectivement dans un spam et dans un ham.

Le problème de cette relation est que lorsque les nombres $N_{spam}$ et $N_{ham}$ sont petits ou nuls, nous ne pouvons pas être sûrs de l'origine du mot (spam ou ham) puisque nous n'avons pas assez de points de mesures. Robinson propose alors de remplacer la probabilité d'avoir le mot dans un spam par la relation suivante :

$$f(m) = \frac{(s \times x) + (n \times p(m))}{s + n}$$

où « x » est la probabilité d'avoir le mot dans un spam lorsque le nombre d'échantillons est nul ou proche de zéro. $N = N_{spam} + N_{ham}$ est le nombre de fois où le mot apparaît dans un mail. Et « s » est la sensibilité lorsque la fréquence du mot est faible. Ici, nous prendrons s = 1 et x = 0,5. Après simplification, f(m) est donnée par la relation :

$$f(m) = \frac{0.5 + N_{spam}}{1 + N_{spam} + N_{ham}}$$

que l'on retrouve dans la fonction `word-proba`. Pour déterminer si un mail est un spam, nous nous servons de la probabilité de chacun des N mots contenus dans le sujet du mail. Ceci est réalisé grâce à la fonction `analyse-spam`. Nous calculons les produits :

$$P_1 = [(1 - f_1) \times (1 - f_2) \times ... \times (1 - f_N)]^{\frac{1}{N}}$$

et

$$Q_1 = [f_1 \times f_2 \times ... \times f_N]^{\frac{1}{N}}$$

La probabilité pour que le mail soit plus un spam ou un ham est alors donnée par :

$$P_{spam} = 1 - P_1 \qquad et \qquad P_{ham} = 1 - Q_1$$

Pour donner une indication sur la nature du mail, nous calculons alors le rapport :

$$S = \frac{P_{spam} - P_{ham}}{P_{spam} + P_{ham}}$$

qui renvoie un nombre compris entre -1 pour un ham et +1 pour un spam. Pour obtenir un nombre entre 0 et 1, nous calculons enfin :

$$P(mail = spam) = \frac{1 + S}{2} = \frac{1 - P_1}{2 - P_1 - Q_1}$$

Tout ceci est résumé dans la fonction `analyse-spam` grâce à la macro `loop`. Nous définissons alors la fonction `spam-proba` qui sera la fonction visible de l'extérieur du package `spam.lisp` et qui utilise les deux méthodes vues précédemment pour déterminer la probabilité qu'un mail soit un spam en analysant une ligne contenue soit dans le sujet du mail soit dans son corps.

Un mail étant considéré comme un spam lorsque la probabilité retournée par `analyse-spam` est supérieure à 0,6. Ceci étant caché par la fonction `is-spam`. De la même manière que pour le module `pop3.lisp`, nous pouvons tester le module `spam.lisp` directement dans la REPL du Lisp :

```
CL-USER> (load "util.lisp")
CL-USER> (load "spam.lisp")
CL-USER> (in-package :spam)
SPAM> (inc-word "bien" :ham)
1
SPAM> (inc-word "viagra" :spam)
1
SPAM> (word-proba "bien")
0.25
SPAM> (word-proba "viagra")
0.75
SPAM> (word-proba "inconnu")
0.5
SPAM> (ham "bien" 100)
100
SPAM> (word-proba "bien")
0.004950495
SPAM> (spam "viagra" 100)
100
```

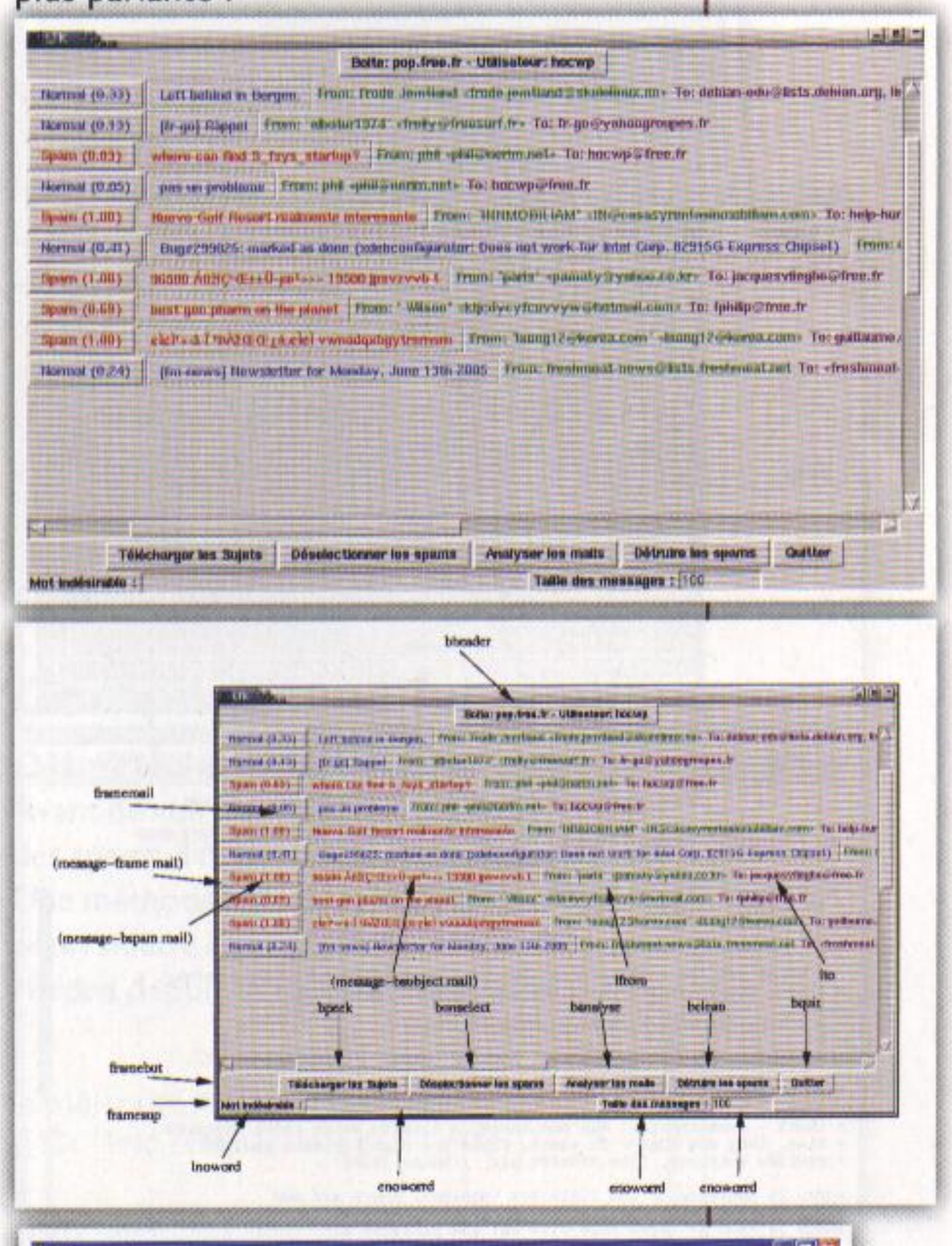
```
SPAM> (word-proba "viagra")
0.9950495
SPAM> (spam-proba "ce mail contient le mot viagra")
0.63625735
SPAM> (is-spam (spam-proba "ce mail contient le mot viagra"))
T
SPAM> (spam-proba "ce mail contient le mot bien")
0.36374256
SPAM> (is-spam (spam-proba "ce mail contient le mot bien"))
NIL
SPAM> (spam-proba "ce mail ne contient aucun mot connu")
0.5
SPAM> (is-spam (spam-proba "ce mail ne contient aucun mot connu"))
NIL
SPAM> (learn-from-line "Enlarge your pen1s" :spam)
NIL
SPAM> (spam-proba "enlarge your pen1s")
0.75
SPAM> (dotimes (i 10) (learn-from-line "Enlarge your pen1s" :spam))
NIL
SPAM> (spam-proba "enlarge your pen1s")
0.9583334
SPAM> (learn-from-line "ceci est un gentil mail inoffensif" :ham)
NIL
SPAM> (spam-proba "ceci est un gentil mail inoffensif")
0.25
SPAM> (dotimes (i 10) (learn-from-line "ceci est un gentil mail
inoffensif" :ham))
NIL
SPAM> (spam-proba "ceci est un gentil mail inoffensif")
0.041666683
SPAM> (is-spam (spam-proba "ceci est un gentil mail inoffensif"))
NIL
SPAM> (is-spam (spam-proba "enlarge your pen1s"))
T
SPAM> (learn-from-file "base-spam.txt" :spam)
NIL
SPAM> (spam-proba "All Windows software for cheap")
0.860036
SPAM> (is-spam (spam-proba "All Windows software for cheap"))
T
SPAM>
```

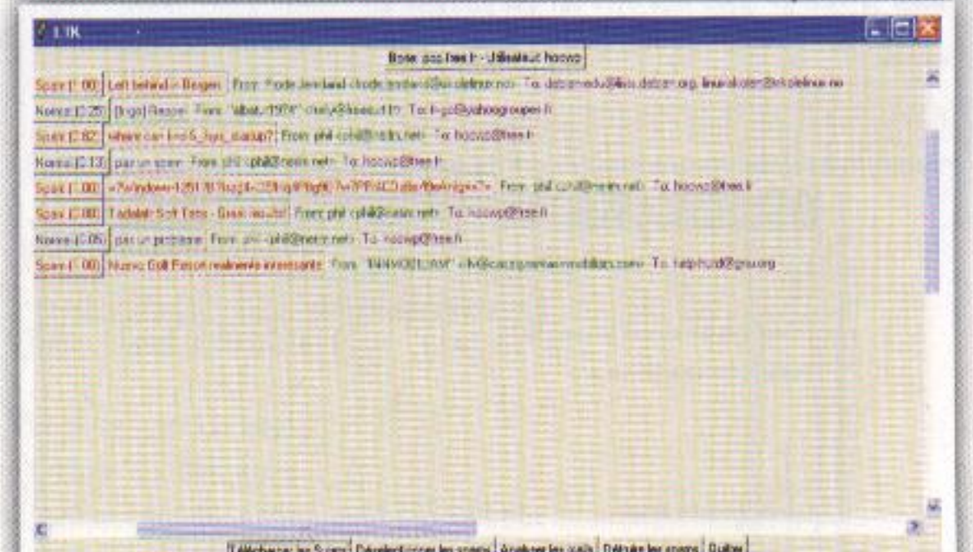
Voilà, tout semble fonctionner correctement : plus un mot est présent dans un spam, plus la probabilité que le mail soit un spam augmente et plus un mot est présent dans un ham, plus la probabilité que le mail soit un spam diminue.

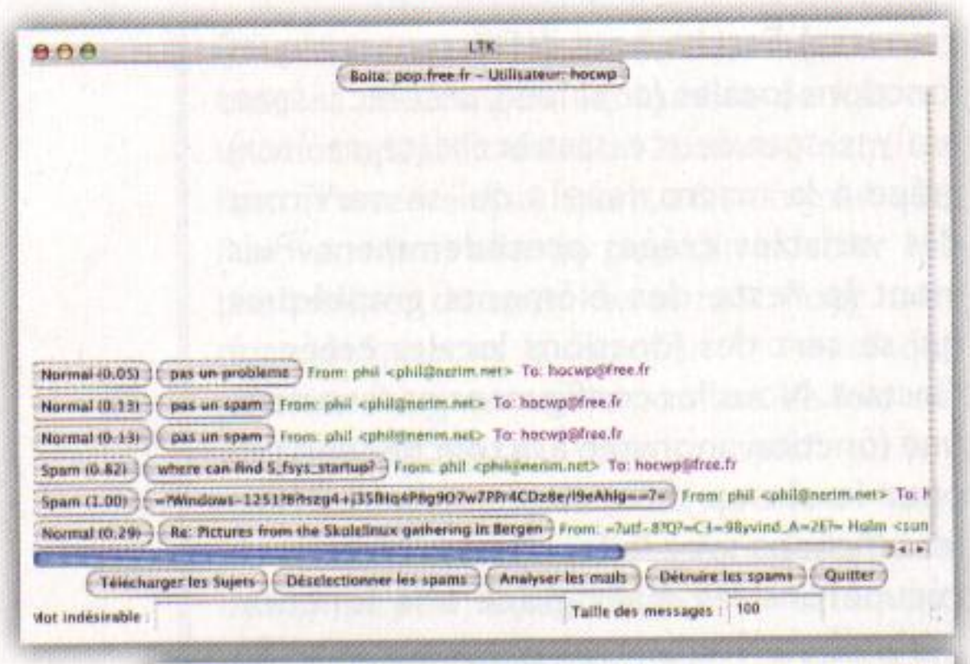
## L'interface de confirmation : ajouter des mots dans le filtre

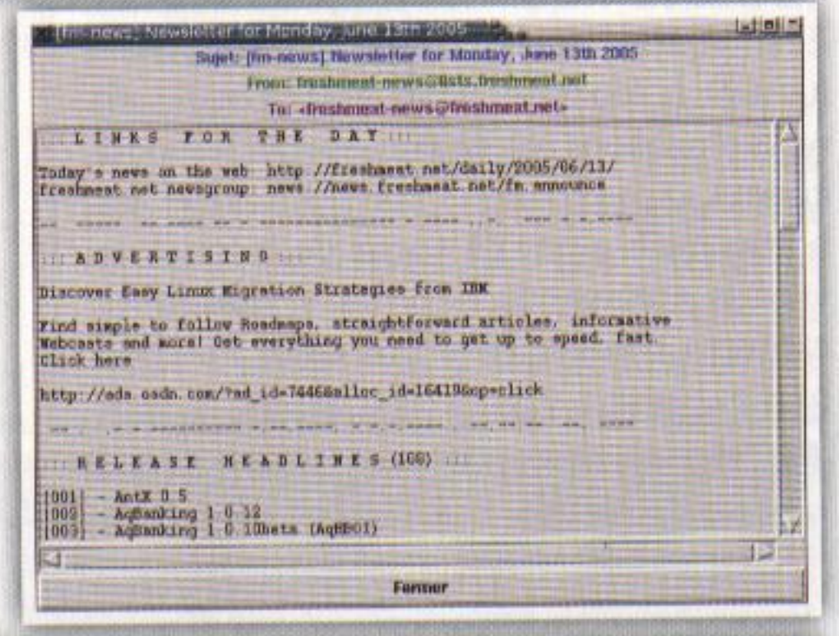
Maintenant que nous savons récupérer nos mails et détecter les spams, il nous reste à faire une interface de confirmation. En effet, il se peut très bien que le filtre laisse passer des spams ou au contraire élimine des mails importants. Pour cela, je vous propose d'utiliser le module `ltk` de Peter Herth [8] qui permet de communiquer avec l'interpréteur de Tcl/Tk et donc de faire des interfaces graphiques portables sans trop de difficultés à condition d'installer Tcl/Tk sur sa machine (quel que soit le système d'exploitation)(Voir code sur le CD "`clmail.lisp`").

Tout d'abord, nous définissons la fonction principale `clmail` : nous lisons le fichier de configuration grâce à la fonction `load-base` qui permet de retrouver les mots déjà trouvés dans des spams et des hams, ainsi que la description des boites mails où trouver les serveurs POP3. Puis nous lançons l'interpréteur Tcl/Tk avec la macro `with-ltk`. Nous créons ensuite une liste contenant tous les mails (`all-mails`) et quelques éléments graphiques (`bheader`, `framemail`, `framebut` et `framesup`). Ensuite, nous définissons quelques fonctions locales (`download`, `unselect-spam`, `analyse-spam`, `delete-spam` et `choise-mailbox`) grâce à la macro `labels` qui se serviront des variables créées précédemment. Puis vient le reste des éléments graphiques qui se sert des fonctions locales créées à l'instant. Nous les configurons et associons une fonction anonyme à la touche [entrée] pour le champ de texte `"enoword"`. Nous les affichons avec la fonction `pack` et nous récupérons les mails grâce à la fonction `peek-all-mails` que nous verrons juste après. Et enfin, nous affichons l'ensemble des mails lus avec la fonction `gui-populate-mails`. Et nous n'oublions pas de sauvegarder les mots connus (`spams/hams/indesirables`) et les boites mails avant de sortir de la fonction avec la commande `save-base`. De jolies captures d'écran rendront cette fonction plus parlante !

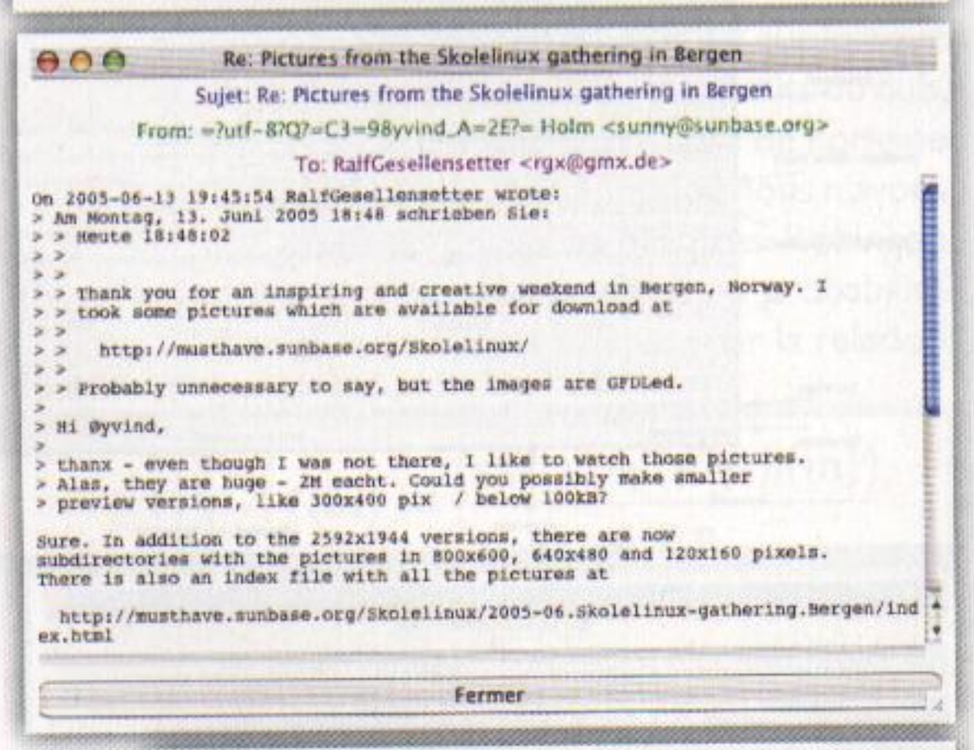


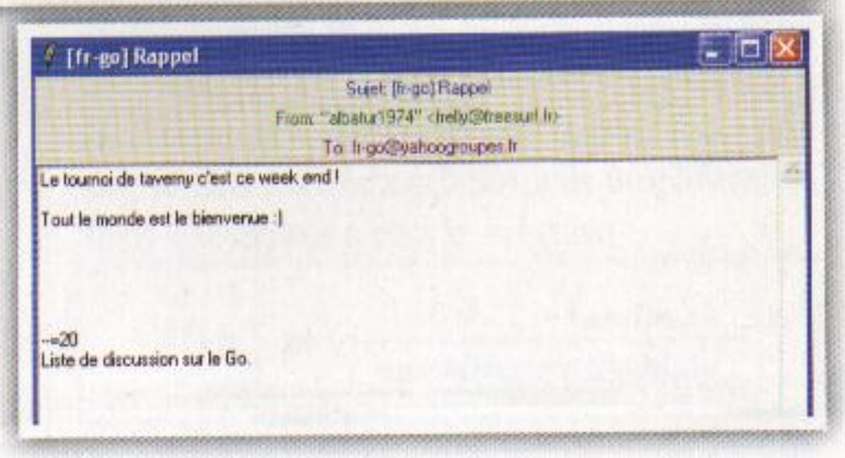










Pour le principe de fonctionnement, nous nous servons du bouton *bheader* pour sélectionner une boite mail (grâce à la fonction `gui-mailbox-choise`) et afficher les informations concernant les boites mail, ainsi que celle concernant l'état du téléchargement des mails. Le bouton *bpeek* permet de télécharger les nouveaux mails. Le bouton *bunselect* permet de mettre à zéro la probabilité de spam de tous les mails. Le bouton *banalyse* permet d'affecter une probabilité de spam à un mail. Le bouton *bclean* permet de détruire les spams sur le serveur POP3 et, au passage, complète la base des mots connus grâce à la fonction `learn-from-line` appliquée au sujet du mail avec le type du mail. Et enfin, le bouton *bquit* permet de quitter l'application. Le champ de texte `"enoword"` servant à rentrer un nouveau mot indésirable. De plus, le corps de chaque mail peut être affiché en cliquant sur le sujet : on fait alors appel à la fonction `gui-display-mail` et on affiche les `*default-body-length*` (10 par défaut) lignes téléchargées en même temps que l'en-tête du mail. Ce nombre pouvant être changé avec l'entrée *Taille des messages*, mais la nouvelle valeur n'est prise en compte que lors du prochain téléchargement.

Vous remarquerez que dans la fonction `gui-populate-mails`, nous avons utilisé une *closure* ou fermeture pour fixer la commande de chaque bouton.

```
(let ((m mail))
  (setf (command (message-bspam mail))
        (lambda ()
          (setf (message-spam-proba m)
                (if (is-spam (message-spam-proba m)) 0 1))
          (gui-set-message-type m))))
```

En effet, ces boutons étant dynamiques, dans le sens où ils changent de texte et de couleurs en fonction de leur état (spam ou ham), nous avons besoin de connaître le mail auquel se rapporte le bouton. Ceci est réalisé en créant la variable `m` qui correspond au mail courant, la fonction anonyme faisant ensuite référence à cette variable. Pour mieux comprendre, dans un langage comme le C, ceci serait équivalent à créer une fonction à la volée où `m` serait une variable statique pointant sur le mail courant. Pour télécharger les mails et les supprimer sur le serveur, nous avons à chaque fois besoin de nous connecter au serveur POP3. Donc, pour simplifier les choses et éviter d'avoir deux fois le même code, nous définissons la macro `with-connection`. Celle-ci entoure le code que l'on veut exécuter avec le code permettant de se connecter au serveur, les tests d'erreur et les mises à jour du bouton *bheader* pour suivre l'évolution de la connexion. Pour voir le fonctionnement de cette macro, voilà une macro plus simple définie sur le même principe :

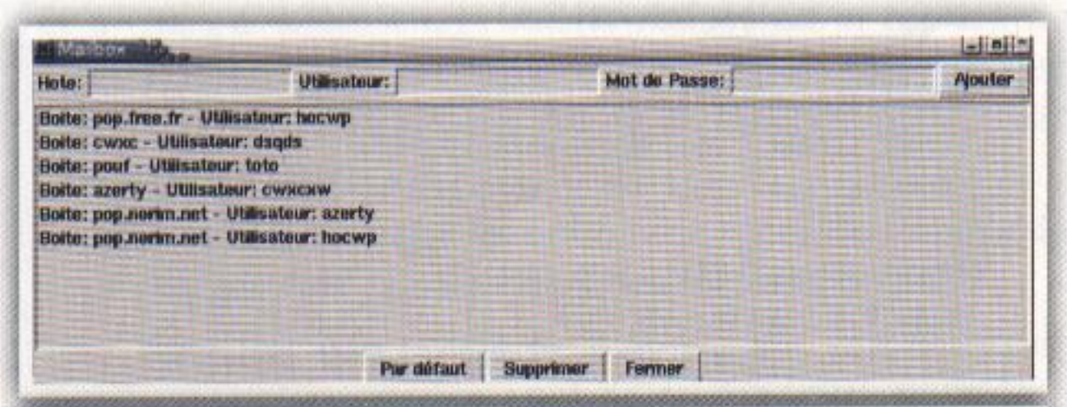
```
CL-USER> (defmacro entoure ((test) &body body)
           (let ((result (gensym)))
             `(let ((,result "Pas de valeur"))
                (format t "Oui, mais fais ça avant~%")
                (if ,test
                    (setf ,result (progn ,@body))
                    (format t "le test est faux~%"))
                (format t "Et puis fais ça après~%")
                ,result)))
CL-USER> (entoure (t) (format t "Je veux renvoyer 10 !~%") 10)
Oui, mais fais ça avant
Je veux renvoyer 10 !
Et puis fais ça après
```

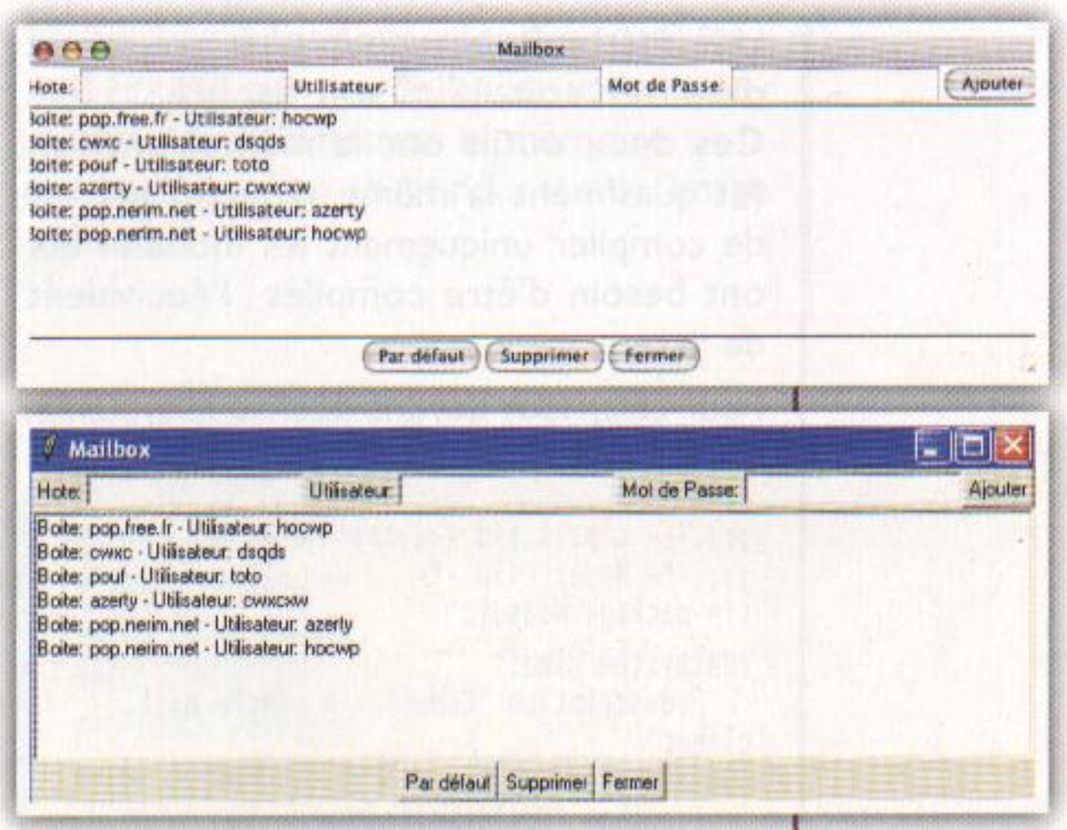
```
10
CL-USER> (entoure (nil) (format t "Je veux renvoyer 10 !~%") 10)
Oui, mais fais ça avant
le test est faux
Et puis fais ça après
"Pas de valeur"
CL-USER> (macroexpand-1 '(entoure (t) (format t "Je veux renvoyer
10 !~%")
                                  10))
(LET ((#:G5501 "Pas de valeur"))
  (FORMAT T "Oui, mais fais ça avant~%")
  (IF T
      (SETF #:G5501 (PROGN
                     (FORMAT T "Je veux renvoyer 10 !~%")
                     10))
      (FORMAT T "le test est faux~%"))
  (FORMAT T "Et puis fais ça après~%")
  #:G5501)
CL-USER>
```

Tout d'abord, nous avons besoin d'une variable qui stockera la valeur de retour de la fonction que l'on entoure. Dans les articles précédents, j'ai parlé des captures de variables. Ici, nous ne voulons pas que la variable `"result"` puisse être vue (ou capturée) de l'extérieur de la macro. Pour cela, nous créons le nom d'une variable locale grâce à la fonction `gensym` qui garantit que le symbole (`#:G5501`) qu'elle retourne est unique. Puis nous créons la variable `#:G5501` et nous lui affectons la valeur `"Pas de valeur"`. Ensuite, nous faisons une première opération en affichant le texte `"Oui, mais fais ça avant"`. Puis, si le test est vrai, nous effectuons le code passé en argument à la macro en affectant le résultat de ce code à la variable `result`, sinon nous affichons le message `"le test est faux"`. Nous affichons, ensuite, le texte `"Et puis fais ça après"` et nous renvoyons la valeur stockée dans la variable `result`.

Nous utilisons ensuite cette macro dans les fonctions `peek-mail` et `clean-spams` sans nous préoccuper de la connexion ou de la gestion des erreurs. Le bouton *bheader* étant mis à jour lors du téléchargement des mails ou de l'effacement des spams. La mise à jour finale indiquant l'état de la boite étant réalisée par la macro.

Sur le même principe que la fonction `clmail`, nous définissons la fonction `gui-mailbox-choise` qui permet de choisir la boite mail à utiliser. Voici quelques captures d'écrans de cette interface, dont une version détaillée pour éviter les trop longs discours.




Par convention dans `util.lisp`, nous avons considéré que la boite par défaut était la première boite de la liste `*mailbox*`. Donc, pour choisir la boite par défaut, nous nous servons de la fonction standard `rotatef` qui permet de permuter des éléments en indiquant leur place, c'est-à-dire, ici, leur position dans la liste `*mailbox*`.

Nous avons aussi besoin de savoir si la fenêtre de choix est déjà ouverte ou non afin de ne pas l'ouvrir deux fois. Pour cela, nous utilisons une fermeture (ou closure) en définissant la fonction `gui-mailbox-choise` à l'intérieur du champ de portée de la variable `open`. Cette variable étant globale pour la fonction `gui-mailbox-choise` mais locale et donc inaccessible pour toutes les autres parties du programme.

## Compiler tout ça : asdf ou mk:defsystem

Avant de voir comment sauvegarder la base des spams, il nous reste à compiler tout ceci. Une méthode fastidieuse et source d'erreur pourrait être de charger chaque module dans l'ordre depuis la REPL ou un script :

```
phil@localhost:[clmail]$ clisp
[1]> (load (compile-file "util.lisp"))
...
[2]> (load (compile-file "net.lisp"))
...
[3]> (load (compile-file "ltk.lisp"))
...
[4]> (load (compile-file "pop3.lisp"))
...
[5]> (load (compile-file "spam.lisp"))
...
[6]> (load (compile-file "clmail.lisp"))
...
[7]> (clmail:clmail)
Nombre de messages : 7
Taille des messages : 42776 octets
```

Une méthode beaucoup plus simple est d'utiliser soit ASDF soit MK:DEFSYSTEM. Ces deux outils ont la même fonction (et quasiment la même syntaxe) qui est de compiler uniquement les modules qui ont besoin d'être compilés : l'équivalent de Make.

Pour cela, nous devons décrire le système. Ce qui donne, avec ASDF, par exemple :

```
;;; --- clmail.asd --- ;;;
;;; -*- Mode: Lisp -*-
(in-package #:asdf)
(defsystem clmail
    :description "CLMail - A simple mail
client"
    :version "0.1"
    :author "Philippe Brochard  "
    :licence "GNU General Public License (GPL)"
    :components ((:file "net")
                 (:file "ltk")
                 (:file "util")
                 (:file "pop3"
                        :depends-on ("net"
"util"))
                 (:file "spam"
                        :depends-on ("util"))
                 (:file "clmail"
                        :depends-on ("pop3"
"ltk" "util" "spam"))))
```

Puis, pour charger le système depuis n'importe quel répertoire, nous définissons le fichier load.lisp qu'il suffira de charger pour avoir tout le programme compilé et chargé automatiquement :

```
;;; --- load.lisp (asdf) --- ;;;
(defparameter *base-dir* (directory-namestring *load-
truename*))
#-ASDF
(load (make-pathname :host (pathname-host *base-dir*)
                     :device (pathname-device *base-dir*)
                     :directory (pathname-directory *base-dir*)
                     :name "asdf" :type "lisp"))
(push *base-dir* asdf:*central-registry*)
(asdf:oos 'asdf:load-op :clmail)
(clmail:clmail nil)
(quit)
```

Tout d'abord, nous définissons la variable *base-dir* qui contient le chemin du répertoire où l'on doit charger le programme. Puis nous testons grâce à une directive conditionnelle #- si le module ASDF est déjà chargé. Cette directive est équivalente aux #ifdef et #ifndef du C. Lorsque le compilateur rencontre le symbole #+asdf, il vérifie dans la liste *features* la présence du symbole 'asdf. Si celui-ci est présent, l'expression suivant la directive est exécutée. A l'inverse, avec un symbole #-asdf, le code sera exécuté si le symbole 'asdf n'apparaît pas dans la liste *features*. Par exemple, voici la liste *features* avec CLisp sous GNU/Linux :

```
[1]> *features*
(:ASDF :SYSCALLS :CLX-ANSI-COMMON-LISP :CLX :REGEXP :CLOS :LOOP :COMPILER
:CLISP
 :ANSI-CL :COMMON-LISP :LISP=CL :INTERPRETER :SOCKETS :GENERIC-STREAMS
 :LOGICAL-PATHNAMES  :SCREEN :FFI :GETTEXT :UNICODE :BASE-CHAR=CHARACTER :
PC386
 :UNIX)
```

Nous indiquons ensuite à ASDF que le répertoire où se trouve load.lisp est un répertoire contenant la définition de systèmes ASDF. Puis, nous compilons et chargeons le programme avec l'abréviation asdf:oos pour « *operate on system* ». Il ne reste plus qu'à exécuter la fonction principale clmail et à faire en sorte de quitter la REPL du Lisp quand le programme est fini.

L'avantage d'utiliser ASDF ou MK:DEFSYSTEM est qu'une fois le programme compilé, vous pouvez placer le répertoire contenant le programme où vous voulez et il ne reste plus qu'à charger load.lisp depuis un script *bash* ou depuis un fichier pif ou ce que vous voulez qui lance une commande *shell*. Et de manière encore plus souple, certaines distributions GNU/Linux utilisent le common-lisp-controller ou asdf-install qui permettent de gérer directement les fichiers ASDF.

## Sauvegarde de la base de spams

Pour finir, chaque fois que nous quittons le programme, la base de mail est oubliée. Il nous faut donc un moyen de la sauvegarder. Pour cela, nous avons (au moins) deux solutions.

La première consiste à créer un fichier de configuration qui sera mis à jour à chaque fois que le programme se termine. Ceci est réalisé grâce aux fonctions load-base et save-base.

```
;;; --- util.lisp (suite) --- ;;;
(defun save-base (&optional (filename *default-rcfile*))
  (with-open-file (stream filename :direction :output :if-exists :
supersede)
    (format stream "(use-package :util)~%")
    (format stream "(use-package :spam)~%")
    (format stream "~2&(setf *default-body-length* ~A)~%"
*default-body-length*)
    (format stream "~2&(setf *mailbox* (list~%")
    (dolist (mailbox *mailbox*)
      (format stream "~&                    (make-mailbox :host ~S
                                :user ~S
                                :password ~S)"
              (mailbox-host mailbox)
              (mailbox-user mailbox)
              (mailbox-password mailbox)))
    (format stream "))")
    (format stream "~2&;; Wrong word part~%")
    (format stream "(setf *wrong-word-list* '~S)~%" *wrong-word-
list*)
    (format stream "~2&;;; Spam part~%")
    (maphash #'(lambda (key val)
                 (format stream "(spam ~S ~A)~%" key val))
             *spam*)
    (format stream "~2&;;; Ham part~%")
    (maphash #'(lambda (key val)
                 (format stream "(ham ~S ~A)~%" key val))
             *ham*)))

(defun load-base (&optional (filename *default-rcfile*))
  (when (probe-file filename)
    (load filename)))
```

La fonction save-base écrit un fichier lisp (~/.clmailrc.lisp) contenant l'état de toutes les variables globales du programme. Par exemple, avec la session précédente du filtre antispam :

```
;;; --- ~/.clmailrc.lisp --- ;;;
(use-package :util)
(use-package :spam)
(setf *default-body-length* 10)
(setf *mailbox* (list
                 (make-mailbox :host "pop.mon_fai.fr"
                               :user "nom_utilisateur"
                               :password "mot_de_passe")
                 (make-mailbox :host "pop.azerty.net"
                               :user "plop"
                               :password "pss")))
;; Wrong word part
(setf *wrong-word-list* '("vlagra" "bite" "sexe" "sex"))
;;; Spam part
(spam "penls" 11)
(spam "your" 11)
(spam "Enlarge" 11)
(spam "viagra" 100)
;;; Ham part
(ham "inoffensif" 11)
(ham "mail" 11)
(ham "gentil" 11)
(ham "un" 11)
(ham "est" 11)
(ham "ceci" 11)
(ham "bien" 100)
```

La fonction load-base ne fait que charger le fichier ~/.clmailrc.lisp après avoir testé son existence. L'avantage de cette méthode est que le fichier de configuration est assez compréhensible pour que l'on puisse le modifier à la main, même si cela n'est pas absolument nécessaire.

La deuxième méthode est terriblement plus efficace, mais n'est pas totalement portable. Celle-ci consiste tout simplement à créer une image de la mémoire en cours d'utilisation au moment de quitter le programme.

Et à charger cette image mémoire lors de la reprise du programme. Cette méthode n'est pas portable puisque chaque implémentation y a été de sa fonction pour créer l'image mémoire.

Et pour ne rien arranger, les options de lignes de commande pour charger l'image ne sont pas les mêmes non plus.

Pour remédier à ce problème, il nous suffit, soit de rester avec une implémentation et dans ce cas, cette deuxième méthode ne tient qu'en _deux_ lignes, soit de faire notre propre fonction portable d'une implémentation à l'autre.

Bien évidement, je vous propose cette deuxième méthode afin de rester le plus général possible.

Pour cela, nous allons nous servir une nouvelle fois des conditions de compilation en testant la version du Lisp que nous utilisons et en appelant la fonction correspondante.

Nous définissons donc la fonction garbage-collect qui permet de lancer le ramasse-miettes pour nettoyer la mémoire et la fonction save-lisp-image pour sauver l'image mémoire :

```
;;; --- util.lisp (suite) --- ;;;
(defun garbage-collect ()
  "Run the garbage collector"
  #+(or CLISP CMU) (ext:gc)
  #+SBCL (sb-ext:gc)
  #+GCL (lisp:gbc t)
  #+ECL (si:gc t)
  #+OPENMCL (ccl:gc)
  #+(or ALLEGRO-CL ALLEGRO-CL-TRIAL) (excl:gc)
  #+LISPWORKS (hcl:normal-gc)
  #+CORMANLISP (cl-user:gc)
  #-(or CLISP CMU SBCL GCL ECL OPENMCL ALLEGRO-CL
        ALLEGRO-CL-TRIAL LISPWORKS CORMANLISP)
  (format t "Sorry, I don't know how to garbage collect with your lisp~%"))

(defun save-lisp-image (filename &rest args)
  "Run the garbage collector and dump a memory image"
  (garbage-collect)
  #+CLISP (apply #'ext:saveinitmem filename args)
  #+CMU (apply #'extensions:save-lisp filename args)
  #+SBCL (apply #'sb-ext:save-lisp-and-die filename args)
  #+GCL (apply #'system:save-system filename args)
  #+ECL (apply #'c:build-program filename args)
  #+OPENMCL (apply #'save-application filename args)
  #+(or ALLEGRO-CL ALLEGRO-CL-TRIAL)
  (apply #'excl:dumplisp :name filename args)
  #+LISPWORKS (apply #'lw:save-image filename args)
  #+CORMANLISP (apply #'cl-user:save-image filename args)
  #-(or CLISP CMU SBCL GCL ECL OPENMCL ALLEGRO-CL
        ALLEGRO-CL-TRIAL LISPWORKS CORMANLISP)
  (format t "Sorry, I don't know how to build an image with your lisp~%"))
```

Et on remplace la fonction *save-base* par :

```
(defun save-base ()
  (save-lisp-image
   #+(or CLISP CMU SBCL OPENMCL ALLEGRO-CL ALLEGRO-CL-TRIAL CORMANLISP)
   "image.core"
   #+GCL "new-gcl"
   #+ECL "new-ecl"
   #+LISPWORKS "new-lispworks"
   #-(or CLISP CMU SBCL GCL ECL OPENMCL ALLEGRO-CL
         ALLEGRO-CL-TRIAL LISPWORKS CORMANLISP)
   (format t "Sorry, I don't know how to save an image with your lisp~%")))
```

L'image produite est plus ou moins lourde, de 2 Mo pour CLisp à 23 Mo pour CMUCL ou SBCL. Elle est un exécutable dans le cas de GCL, ECL et Lispworks. Elle est aussi lourde parce qu'elle contient tous les symboles du Lisp (symboles et fonctions), ainsi que leurs documentations.

Ensuite, il ne reste plus qu'à redémarrer le Lisp avec l'image produite précédemment (ce qui varie encore d'une implémentation

à l'autre) en lançant la fonction `clmail` au démarrage :

```
$ clisp -M image.core  -x "(clmail:clmail)"
$ cmucl -core image.core -eval "(clmail:clmail)"
$ sbcl --core image.core --eval "(clmail:clmail)"
$ ./new-gcl -eval "(clmail:clmail)"
$ ./new-ecl -eval "(clmail:clmail)"
$ openmcl -I image.core -e "(clmail:clmail)"
$ acl -I image.core -e "(clmail:clmail)"
$ ./new-lispworks
```

Si vous voulez utiliser cette méthode, il sera judicieux de faire un test à l'installation du programme et de créer un script shell contenant la ligne de commande voulue suivant l'implémentation utilisée.

De plus, l'image devra être sauvée dans un endroit où l'utilisateur a les droits en écriture (donc ne pas oublier de fixer les droit sur l'image lors de l'installation).

Cette méthode peut paraître longue, mais à l'usage, le chargement de la base est extrêmement rapide quelle que soit sa taille.

De plus, une fois ces fonctions écrites, cette méthode devient tout à fait portable. Par contre, si l'utilisateur veut modifier la base, il doit accéder directement aux différentes variables depuis la REPL du Lisp.

## Conclusion

Voilà, ce petit programme (550 lignes d'après Sloccount, sans les bibliothèques `net.lisp`, `ltk.lisp`, `asdf.lisp` et `defsystem.lisp`) est une application complète avec procédure d'installation et sauvegarde de la base des spams !

J'espère que cette interface de « *Power User* » vous sera utile. Après quinze jours d'utilisation, j'ai une détection de spams « à vue de nez » de l'ordre de 80%, les 20% restant étant dus à de nouveaux mails ou à des sujets peu explicites.

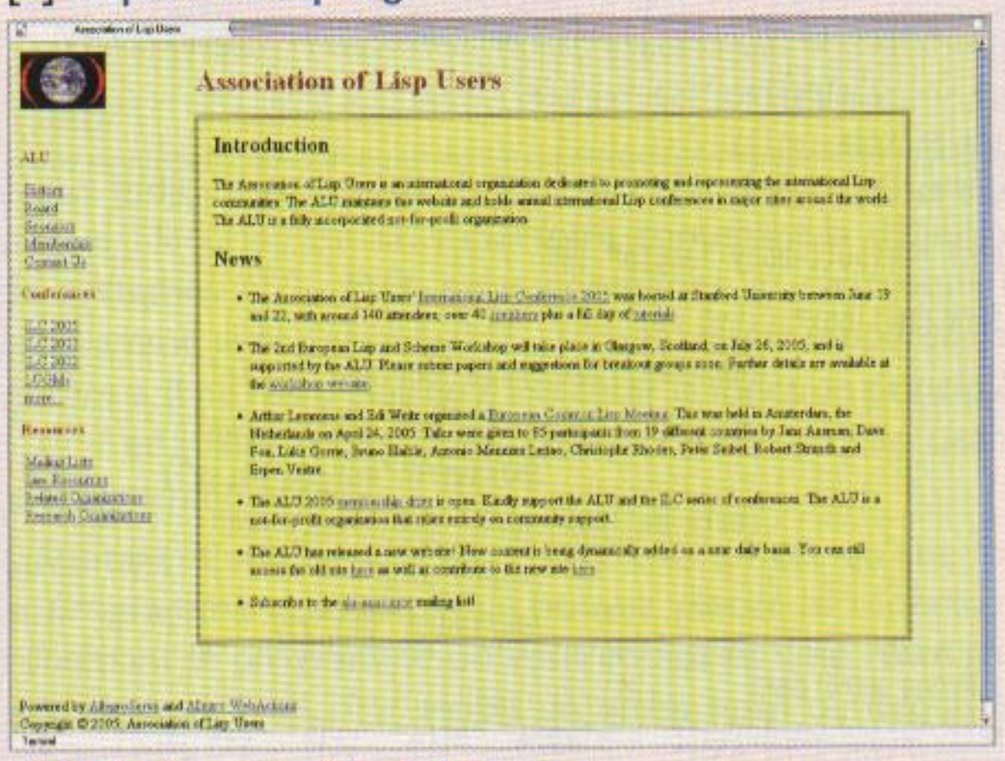
Enfin, une dernière remarque, testez ce programme sur une boite mail peu importante dans un premier temps parce que «*[...] This program is distributed in the hope that it will be useful,but WITHOUT ANY WARRANTY; [...]*», dixit la GPL. Bon hack !

**Philippe Brochard,**

**hocwp@free.fr**

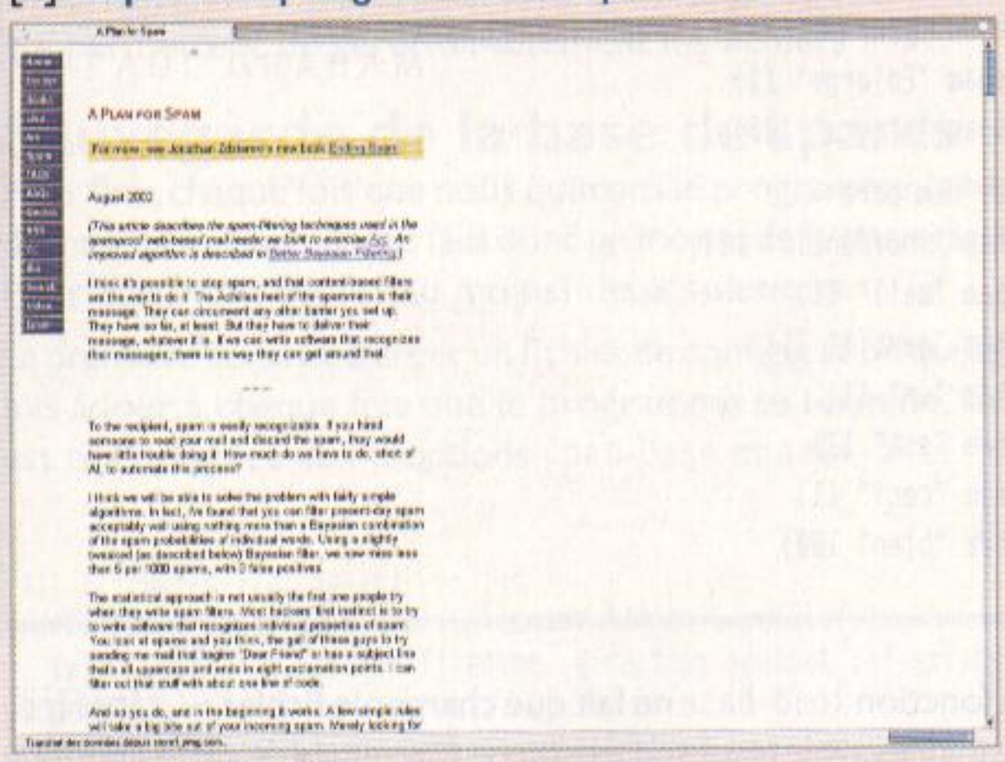
## LIENS

[1] http://www.lisp.org/




[2] http://www.weitz.de/cl-ppcre/

[3] http://www.paulgraham.com/spam.html

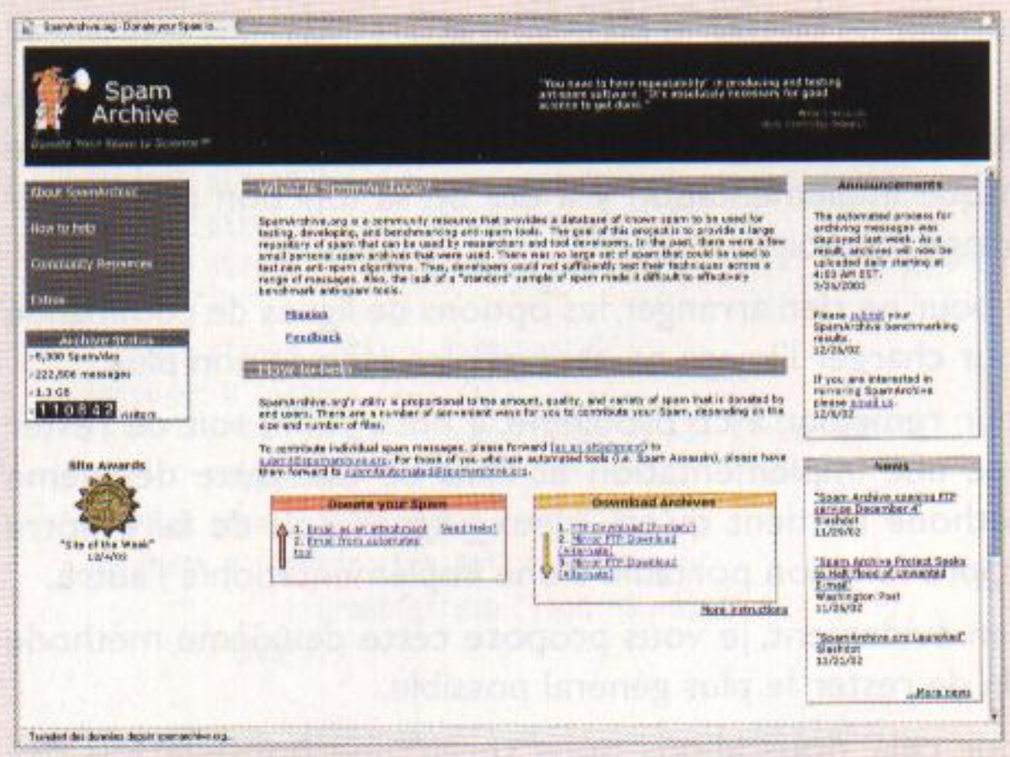



[4] http://spamassassin.apache.org/

[5] http://bogofilter.sourceforge.net/

[6] http://radio.weblogs.com/0101454/stories/2002/09/16/spamDetection.html

[7] http://spamarchive.org/




[8] http://www.peter-herth.de/ltk/

# → Perles de Mongueurs

Sébastien Aperghis-Tramoni

## Chercher sur le CPAN

*There is more than one way to do it.*

Cette maxime bien connue s'applique sans surprise également au CPAN, le *Comprehensive Perl Archive Network*, cœur de la communauté Perl. Explorons donc les différentes manières de chercher (et de trouver) des informations sur le CPAN.

## Search CPAN

C'est le plus ancien, le plus connu et évidemment le plus utilisé. Développé par Graham Barr, et mis en place par Robert Spier et Ask Bjørn Hansen, il dispose depuis peu d'un serveur miroir en Europe afin de répartir la charge d'utilisation. Le code qui fait tourner Search CPAN se nomme TUCS (*The Ultimate CPAN Search*) et n'est malheureusement pas disponible. Il comporte un moteur de recherche (moyennement efficace), ainsi qu'un accès aux distributions par catégorie, par nom de distribution ou par nom d'auteur. Il offre aussi la liste des distributions récemment déposées sur le CPAN. Les distributions disposent d'un URL permanent de la forme **http://search.cpan.org/dist/Class-DBI/**, d'où on peut accéder aux sources actuelles et antérieures de la distribution, ainsi qu'à la documentation agréablement mise en forme. Sont aussi affichés les résumés d'informations de *CPAN Testers* et *CPAN Ratings* ainsi que les liens vers les pages détaillées correspondantes, et enfin les liens vers *CPAN Forum*, vers le gestionnaire de bugs RT du CPAN et vers le répertoire des archives de l'auteur sur BackPAN (**http://backpan.perl.org/**).

Particularité de ce site, deux outils d'examen des sources sont disponibles, un **grep** et un **diff**. Le répertoire d'un auteur est quant à lui accessible par une URL de la forme **http://search.cpan.org/~autrijus/** pour, par exemple, le répertoire (très fourni) d'Autrijus Tang.

## CPAN Suggest

*CPAN Suggest* n'est qu'une interface à *Search CPAN* inspirée de *Google Suggest* et adaptée pour le CPAN. Cela permet de chercher et de trouver plus rapidement des modules, les suggestions s'affinant au fur et à mesure de la frappe.

## CPAN Search Lite

Cet autre site a été développé par Randy Kobes pour répondre au besoin de disposer d'un moteur de recherche dont les sources soient disponibles. Comme sur *Search CPAN*, un moteur de recherche limité et des index (catégories, noms d'auteur, noms de module, noms de distribution) permettent de chercher les modules et distributions. La page des distributions récemment déposées sur le CPAN dispose en plus d'un flux RSS pour ceux qui aiment être tenus au courant.

Là aussi, les distributions sont accessibles par un URL permanente de type **http://kobesearch.cpan.org/dist/CPAN-Search-Lite**. La présentation est similaire à celle de *Search CPAN*, mais présente quelques différences, notamment en offrant les liens directs vers les modules requis par une distribution et des liens vers les dépôts de paquets PPM pour les utilisateurs d'ActivePerl. La documentation est aussi disponible et mise en forme, quoique avec une présentation pour le moment un peu moins agréable. En contrepartie, les sources des programmes sont disponibles depuis peu avec une coloration syntaxique bien utile. Et ici également le répertoire d'un auteur est accessible par un URL telle que **http://cpan.uwinnipeg.ca/~rkobes/**. Toutefois, et contrairement à *Search CPAN*, seule la dernière version de chaque distribution est disponible sur *CPAN Search Lite*, car il s'appuie sur un miroir de type Mini-CPAN alors que *Search CPAN* est un miroir complet. Randy Kobes a de plus récemment doté *CPAN Search Lite* d'une nouvelle fonctionnalité, l'installation des distributions via `PAR::WebStart`. Vous l'aurez deviné, `PAR::WebStart` est l'équivalent pour Perl de Java WebStart, mais en utilisant des archives PAR.

Dernier point sympathique, l'interface de *CPAN Search Lite* est disponible en plusieurs langues, pour le moment anglais, français, espagnol, allemand, portugais et italien, bientôt en russe, bulgare et moldave, pourquoi pas dans d'autres langues encore demain. Tout ceci fait de *CPAN Search Lite* la base idéale pour créer un serveur local de recherche et de documentation, par exemple à l'échelle d'une université ou d'une entreprise.

Code source de *CPAN Search Lite* : **http://cpan.uwinnipeg.ca/dist/CPAN-Search-Lite**

## Gonzui

Principalement développé par Tatsuhiko Miyagawa, Gonzui est un moteur de recherche spécifique au code source. Son but est d'offrir aux développeurs de Logiciels libres un moyen rapide et efficace pour trouver des morceaux de code au sein d'archives aussi variées que celles disponibles sur le CPAN. Écrit en Ruby, il supporte déjà plusieurs langages dont Perl, C, JavaScript, Python, Ruby, PHP et OCaml, le support d'autres langages étant en cours de test.

Contrairement aux autres sites présentés ici qui sont plutôt axés sur la documentation et les métainformations des distributions, Gonzui n'offre qu'une fonction de recherche de code, mais

l'affichage des résultats est assez impressionnant, offrant un code source avec coloration de syntaxe et surlignement automatique de code. Depuis qu'il a été rendu public, il s'avère ainsi très utile aux mainteneurs de Perl pour déterminer si certaines parties obscures de l'API sont ou non utilisées, et ainsi enclencher la tronçonneuse quand c'est possible.

Code source de Gonzui : **http://gonzui.sourceforge.net/**

## CPAN Forum

Disponible depuis février 2005, *CPAN Forum* a été développé par Gabor Szabo afin d'offrir un espace de dialogue par le biais d'un forum web entre l'auteur d'un module et ses utilisateurs quand ils ne disposent pas d'une liste de diffusion.

Suivant la tradition des URL permanents, la page d'une distribution est accessible, vous l'aurez deviné, à **http://www.cpanforum.com/dist/WWW-Mechanize** pour le module `WWW::Mechanize` par exemple. Cette page comprend les liens vers *Search CPAN*, *CPAN Search Lite*, RT CPAN, *CPAN Ratings* et *CPAN Testers*.

On peut s'inscrire sur le site et demander à être prévenu dès qu'un message est ajouté à l'un des forums, typiquement pour surveiller les modules qu'on a écrits ou qu'on utilise intensivement. Sont aussi disponibles un flux RSS par distribution et un flux RSS global qui donnent les 10 derniers messages postés.

## AnnoCPAN

Nouvel arrivé (le site a été rendu public fin juin), *AnnoCPAN* est un site ambitieux développé par Ivan Tubert-Brohman qui permet aux utilisateurs d'annoter en ligne la documentation des modules du CPAN, avec une gestion assez avancée des versions. Cela ressemble un peu à la documentation en ligne de MySQL et PostgreSQL par exemple, mais en permettant une annotation au paragraphe plutôt qu'au chapitre. Une autre différence par rapport à ces deux sites est que les annotations peuvent aussi être supprimées si l'auteur intègre les remarques à la documentation ou corrige les points signalés, évitant ainsi une pollution par accumulation de notes obsolètes.

Respectant la tradition des autres sites web relatifs au CPAN, les distributions sont ici aussi accessibles par des URL de la forme **http://www.annocpan.org/dist/Class-DBI/**, qui outre les liens habituels, permettent de visualiser la documentation annotée pour chaque version disponible. Idem pour les URL d'auteurs, **http://www.annocpan.org/~saperi/**, qui offrent en plus un flux RSS permettant de connaître les dernières notes ajoutées à la documentation d'un module de cet auteur. Un autre flux RSS global est disponible sur la page principale.

# Les nouvelles de la communauté Perl

## Google Summer of Code

Le *Google Summer of Code* (l'été de programmation de Google) est un programme destiné à ouvrir des étudiants au monde des Logiciels libres, en offrant un prix de 4 500 USD à tout étudiant qui pourrait compléter un projet avant la fin de l'été. Pour encadrer tout ceci, Google fait participer plusieurs organisations du Logiciel libre telles que l'Apache Software Foundation, la Gnome Foundation, KDE, OpenOffice.org, Samba... et la Perl Foundation. Celle-ci a lancé un appel à projets, dont huit ont été retenus. Parmi ceux-ci, saluons le projet du mongueur français Alexandre Buisse d'écrire un *garbage collector* plus efficace pour la machine virtuelle Parrot, dont il a rapidement présenté les grandes lignes lors d'une présentation éclair aux Journées Perl 2005. Nommé GMC pour *Generational Mark & Compact*, son projet de garbage collector se base sur le GC actuellement intégré à Parrot ainsi que sur celui qu'avait commencé à programmer Leopold Tötsch. Alexandre sera d'ailleurs supervisé par Leopold pour ce projet.

Pour résumer brièvement, Alexandre compte implanter un algorithme générationnel à *mark-and-sweep* permettant la copie, et qui soit de plus compatible avec les threads. Le principe de cet algorithme est de classer les objets en fonction de leur âge : plus un objet a une adresse mémoire basse, plus il est vieux. On classe ensuite les objets en *générations* qui forment une liste chaînée d'objets. Ainsi, on pourra collecter d'abord les objets les plus jeunes qui, statistiquement, ont une plus courte durée de vie. En posant l'hypothèse (vraie tant qu'on reste dans un langage fonctionnel) que les pointeurs vont tous des nouveaux objets vers les anciens, on peut alors les marquer en une seule passe, en partant des plus récents, à la manière d'un crible d'Ératosthène. Dès qu'une génération a été analysée, tous les objets non marqués peuvent être détruits et les autres viennent combler les « trous » ainsi créés. Il y a bien sûr d'autres subtilités à prendre en compte car cette hypothèse peut être violée dans certains cas qu'il faut donc prendre en compte et il faut également pouvoir collecter les objets qui se pointent mutuellement. Alexandre a néanmoins ses idées là-dessus et a déjà commencé à publier des documents détaillés sur le fonctionnement et l'implémentation de GMC.

### LIENS

Google Summer of Code :
**http://code.google.com/summerofcode.html**
Perl Foundation : **http://perlfoundation.org/**
Perl Foundation: Google Summer of Code Projects Announced : **http://perlfoundation.org/news/2005/googlesoc2.html**
New generational GC Scheme : **http://www.mail-archive.com/perl6-internals@perl.org/msg28393.html**
Page personnelle d'Alexandre :
**http://perso.ens-lyon.fr/alexandre.buisse/**
Nattfodd Summer of Code Blog (le *blog* d'Alexandre Buisse) : **http://nattfodd.blogspot.com/**
Interview d'Alexandre Buisse par Leon Brocard :
**http://use.perl.org/~acme/journal/25627**

## À vous !

Envoyez vos perles à perles@mongueurs.net, elles seront peut-être publiées dans un prochain numéro de Linux Magazine.

**Sébastien Aperghis-Tramoni,**

**sebastien@aperghis.net,**
**Marseille.pm**

# → Tuning de code
## optimisation d'un filtre

Yann Guidon

EN DEUX MOTS **Cela commence par « Est-ce que ce code simple est aussi rapide que possible ? » et nous embarque dans « Quelle est la méthode la plus performante ? ».**

Ce quatrième intermède dans la série sur la compression de données n'est pas seulement un exercice d'optimisation ou une caractérisation du comportement d'un noyau particulier (un vieux 2.4.19pre7), nous dégagerons aussi quelques points permettant d'écrire des programmes un peu plus efficaces.

## 1) La question

Dans le premier article d'introduction à la compression des signaux (GLMF numéro 73, juin 2005), je mentionne qu'on peut traiter des fichiers au moyen de programmes « filtres » qui peuvent avoir la forme générale suivante :

```
typedef signed short int SAMPLE;
SAMPLE droite, gauche;
FILE entree, sortie;

entree = fopen(nom_fichier_entree, "rb");
sortie = fopen(nom_fichier_sortie, "wb");

while ( fread(&gauche, 1, sizeof(SAMPLE), entree)
      + fread(&droite, 1, sizeof(SAMPLE), entree)
        == sizeof(SAMPLE) * 2) {

/* traitement des échantillons ici */

 fwrite(&gauche, 1, sizeof(SAMPLE), sortie);
 fwrite(&droite, 1, sizeof(SAMPLE), sortie);
}

fclose(entree);
fclose(sortie);
```

Cependant, les fichiers à traiter sont *très* gros et les 618Mo de notre fichier de référence ne sont qu'un avant-goût de certaines applications envisagées pour la suite de la série sur la compression. Une petite modification du code source pourrait entraîner une variation de plusieurs secondes sur le temps total d'exécution. L'appel aux fonctions `fread()` et `fwrite()` pour *chaque* échantillon me semble être un premier point à éviter. J'avais écrit le code ci-dessus dans un souci de concision et de simplicité, mais est-ce adapté aux situations plus tendues, telles que dans une application « temps réel » comme l'enregistrement ou la lecture d'un fichier *très* long ? L'idée de cet article est d'examiner le comportement du noyau et de coder un autre filtre plus performant, même s'il doit être plus compliqué. Cependant, comme le résultat doit être exécutable sur presque n'importe quel système Linux, l'éventail des techniques possibles est restreint. En raison de cette généralité, utiliser le mot « optimisation » dans le titre de cet article serait presque un abus de langage s'il n'y avait pas toutes les mesures. Tout bon codeur sait qu'optimiser sans mesurer conduit à de pires résultats qu'au départ, et ce qui suit en est un exemple flagrant.

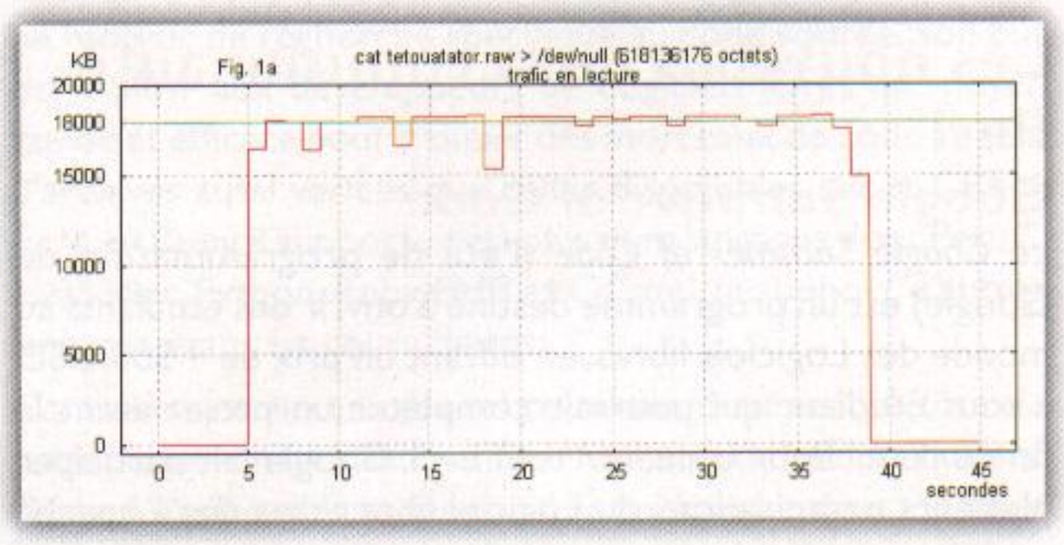
## 2) Les premières mesures

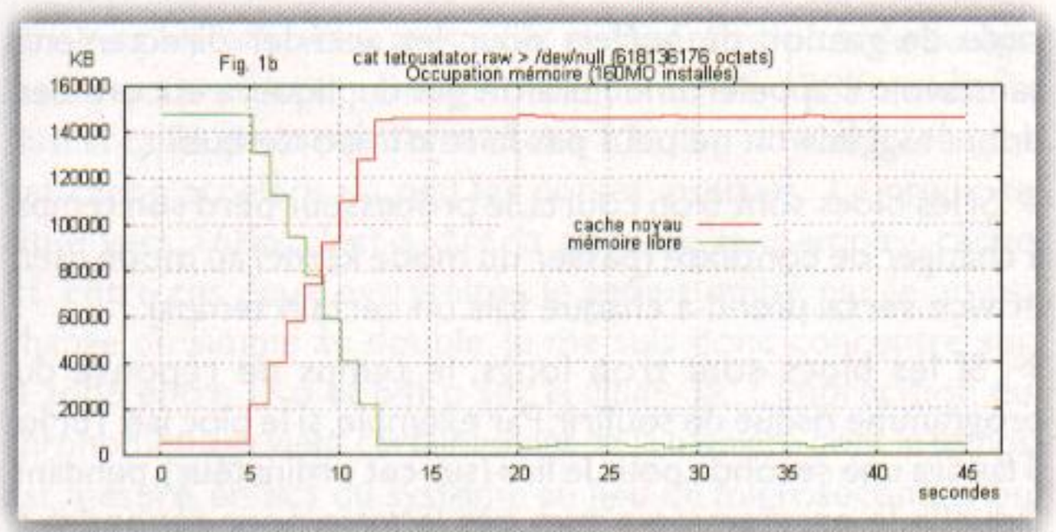
Pour commencer, intéressons-nous aux cas les plus simples : la lecture et l'écriture des données sur le disque. Tout le reste n'est qu'une combinaison des effets de ces deux opérations.

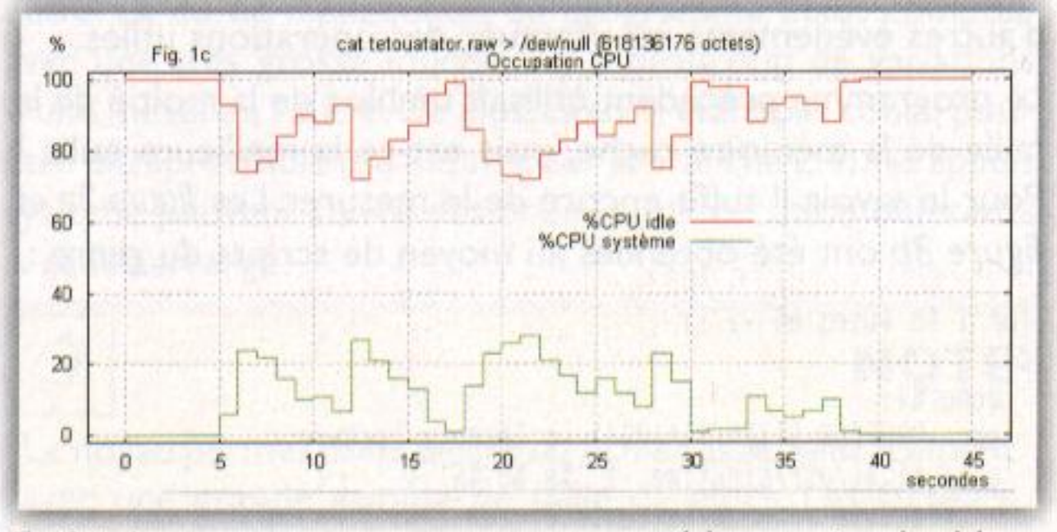
### 2.1) Lecture

La première mesure concerne la lecture de notre fichier de référence (fourni avec GLMF numéro 73 de juin 2005, mais tout fichier de 618136176 Ko fait aussi bien l'affaire). La commande `cat tetouatator.raw > /dev/null` est expédiée en 34 secondes. La *figure 1a* montre que le débit moyen tourne autour de 18Mo/s (618136176/18000 Ko/s =~ 34s : le compte y est). Un paramètre important pour ce type de mesure est la mémoire installée. Le noyau Linux utilise presque toute la RAM disponible pour y loger les derniers morceaux de fichiers accédés. Ainsi, tout accès ultérieur sera presque instantané. Cette mémoire cache en lecture peut interférer avec les mesures : s'il reste un petit bout de fichier en RAM, l'exécution suivante sera anormalement plus courte. Le moyen d'éviter cela (pour des mesures répétitives) est d'opérer sur des fichiers au moins deux fois plus gros que la mémoire installée dans l'ordinateur. Ici, il dispose de 160Mo de RAM et `tetouatator.raw` occupe 618Mo, ce qui ne pose pas de problème (618/160=3,86 > 2).

La *figure 1b* montre l'état de la cache de lecture. Au rythme de 18Mo par seconde, il faut 8 secondes pour épuiser toute la RAM libre. Encore 8 secondes plus tard, toute la mémoire cache a été réécrite. On observe même une légère oscillation sur une période d'environ 8 secondes. La *figure 1c* présente la charge du processeur durant la lecture. Le disque dur tourne à plein régime mais grâce au mode DMA, le processeur n'est occupé qu'au quart de ses possibilités dans le pire des cas. On observe aussi la même oscillation que sur la *figure 1b*, mais amplifiée. Je n'ai aucune explication sur la raison de telles variations.






En fait, le processeur est surtout occupé à recopier des blocs de données en RAM :

- Une fois qu'un bloc est lu en provenance du disque, il est recopié de la cache du noyau vers l'espace utilisateur (*userland*) du programme `cat`.
- `cat` envoie ce bloc (par `write()`) en sortie standard, où le *shell* le lit : c'est au moins une copie supplémentaire, de userland(cat) à userland(shell), qui passe probablement aussi par le noyau.
- Le shell va écrire ce bloc vers `/dev/null`. Il y a fort à parier qu'une copie supplémentaire de userland(shell) à kernel est aussi effectuée par `write()`, qui ignore probablement que le bloc sera ignoré.

Il y a plusieurs spéculations, mais il ressort que passer par le shell induirait une occupation CPU plus grande (à cause de la sur-duplication des données) que si on avait écrit un simple programme en C pour lancer uniquement `read()`.

Il reste cependant assez de temps CPU pour faire d'autres choses à côté (c'est un pauvre Celeron Coppermine rev.6 à 550MHz pour donner une vague idée).

**NOTE**

Permettre à un programme de lire ses données dans `stdin` est une fonction très pratique et facile à coder. Cela évite de perdre du temps dans les cas où le flux à traiter doit passer par plusieurs filtres, obligeant sinon à écrire le flux transformé sur le disque avant de le relire.

Toutefois, si la vitesse importe vraiment et si le programme reçoit les données directement d'un fichier sans passer par d'autres programmes à travers un *pipe* du shell, il vaut mieux effectuer « soi-même » la lecture pour réduire les duplications en RAM (et encore, on ne parle pas de `fread()`/`fwrite()`).

Un programme doit donc utiliser ces deux techniques pour être efficace dans les deux situations.

## 2.2) Réduction des copies

Pour vérifier que les duplications peuvent être réduites, il suffit d'écrire un petit programme, tel que `lecture.c`, ignorant simplement le résultat de la lecture.

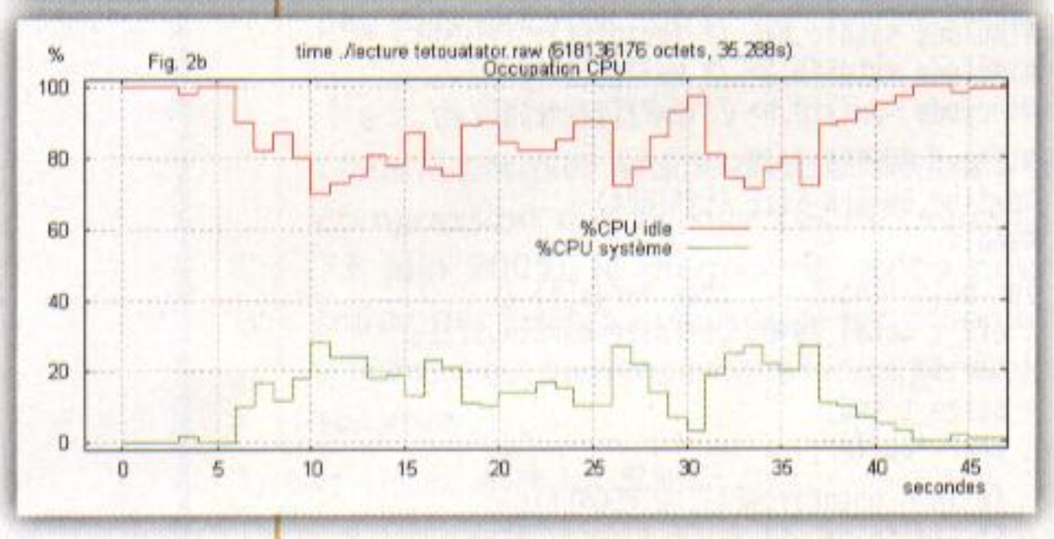
```
/* fichier lecture.c
créé par Yann GUIDON (whygee@f-cpu.org)
Fri Jul 29 10:46:21 CEST 2005

 fonction :
   lire un fichier le plus vite possible

 compilation :
   gcc -Wall -g -o lecture lecture.c

 invocation :
   lecture nom_du_fichier
*/

#define _FILE_OFFSET_BITS 64
#define _LARGEFILE_SOURCE
#define _LARGEFILE64_SOURCE

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>  /* open() */
#include <errno.h>  /* perror() */
#include <stdio.h>  /* fprintf(), fflush() */
#include <stdlib.h> /* malloc() */
#include <unistd.h> /* exit(), read() */

#ifndef BUFFER_SIZE
#define BUFFER_SIZE (64*1024)
#endif

int main (int argc, char *argv[]) {
  off_t total_lu=0, buffsize=BUFFER_SIZE;
  int fd_in;
  ssize_t t=0;
  char *buffer;

  fd_in = open(argv[1], O_RDONLY);
  if (fd_in == -1) {
    perror("Erreur à l'ouverture de la source");
    exit(EXIT_FAILURE);
  }
  buffer=malloc(buffsize);
  if (buffer == NULL) {
    perror("Erreur de malloc() ");
    exit(EXIT_FAILURE);
  }
  while ( (t = read(fd_in, buffer, buffsize)) > 0 ) {
    total_lu+=t;
    fprintf(stderr,"%lld%c", total_lu, 0xD);
    fflush(NULL);
  }
  if (t < 0) {
    perror("Erreur de lecture ");
    exit(EXIT_FAILURE);
  }
  exit(EXIT_SUCCESS);
}
```

**NOTE**

L'objectif initial était de tester l'option `READ_O_DIRECT` mais celle-ci ne fonctionne pas sur les fichiers normaux (sur ce système-là, en tout cas). Voir GLMF numéro 68 de janvier 2005 pour une discussion à ce sujet.

Sans l'option `READ_O_DIRECT`, on peut tout de même observer la différence d'utilisation du processeur. Ici, la taille du tampon de

lecture a été choisie pour être la moitié de la taille de la mémoire cache L2, afin de réduire l'impact de la duplication du tampon vers userland. La figure *figure 2a* montre que le débit et le profil sont tout à fait comparables. La *figure 2b* suggère qu'il n'y a pas de différence dans l'occupation totale du processeur, malgré une forme un peu différente.


On remarque aussi que le temps d'exécution augmente d'une seconde. L'hypothèse sur l'impact des copies internes n'est donc pas vérifiée ici.

## 2.3) Importance de la taille du tampon

Voici encore un sujet de discussion interminable. On peut s'attendre à ce que le système (librairies et noyau) se débrouille correctement dans une grande variété de situations pour réconcilier tout le monde. Pourtant, il doit bien y avoir un point de fonctionnement meilleur que les autres, même s'il dépend du système ?

Le code du filtre original repose sur la librairie d'entrée/sortie standard, qui s'occupe de gérer des tampons avant de les envoyer au noyau. Mais cela ajoute des instructions qui ne contribuent pas directement à remplir la fonction du programme. En gérant nous-mêmes nos tampons, nous nous assurons qu'il n'y a pas trop d'instructions superflues.

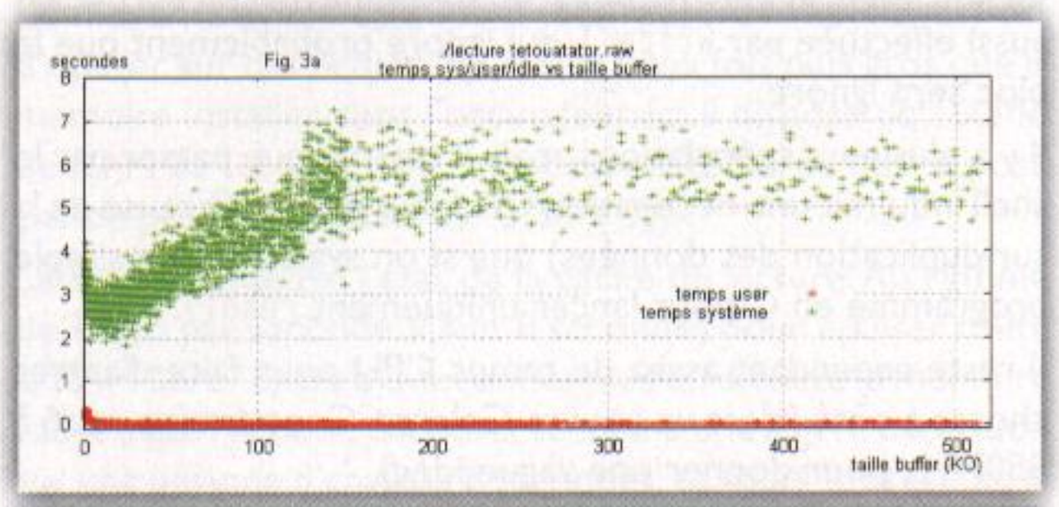
De toute façon, si vous lisez cet article, c'est que vous n'avez pas l'intention de passer par les fonctions `fwrite()` et `fread()`. Les programmes que nous codons devant manipuler des gigaoctets de données, on peut faire l'effort de coder notre propre code de gestion de *buffers* pour les accéder directement, sans avoir à appeler une librairie qui dupliquera encore des données. Mais on ne peut pas faire n'importe quoi :

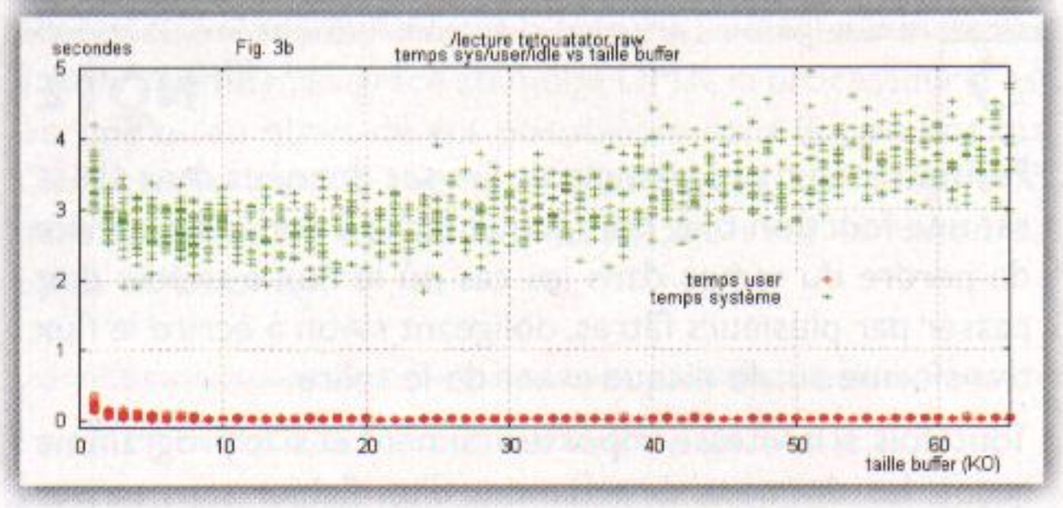
- Si les blocs sont trop courts, le processeur perd son temps à changer de contexte (passer du mode *kernel* au mode *user*, et vice versa, prend à chaque fois un certain temps).
- Si les blocs sont trop longs, le temps de réponse du programme risque de souffrir. Par exemple, si le bloc fait 16Mo, il faudra une seconde pour le lire (sur cet ordinateur), pendant laquelle le programme n'aura pas la main pour répondre à d'autres événements ou effectuer des opérations utiles.

Le programme précédent utilisait un bloc de la moitié de la taille de la mémoire cache, mais est-ce la meilleure taille ? Pour le savoir, il suffit encore de le mesurer. Les *figure 3a* et *figure 3b* ont été obtenues au moyen de scripts du genre :

```
for i in $(seq 64 -1 1)
do
  echo $i:
  gcc -DBUFFER_SIZE=($i*1024) -o lecture lecture.c
  echo $i $(/usr/bin/time -f '%e %U %S' \
    ./lecture /common/tetouatator.raw 2>&1 ) >> bufsz.2.out
done
```

La boucle est dans l'ordre décroissant pour réduire autant que possible les erreurs de mesure dues au démarrage du script et je voudrais savoir exactement ce qui se passe à 1Ko. J'ai aussi enlevé le code d'affichage de la progression, aussi bien pour ne pas perdre de temps que pour ne pas gonfler le fichier de sortie avec des nombres sans intérêt. Avec plus de 1900 points sur la *figure 3a* et 35s par point, je vous laisse calculer le temps qu'il a fallu pour obtenir un des graphes les plus intéressants de ces pages, puisqu'on y voit enfin une *singularité*.




Le temps total ne varie quasiment pas (34,5s, plus ou moins un quart de seconde) car c'est l'accès au disque qui est prépondérant. De plus, les mécanismes de préchargement spéculatif dans le noyau et le disque dur masquent presque complètement l'influence des blocs très courts. On voit que ce n'est pas le temps *user* qui varie, mais le temps *systeme*, à part pour les très petits blocs. Les très gros blocs ne semblent

pas les plus efficaces, puisqu'ils prennent le plus de temps à traiter. La situation change brusquement à 128Ko, c'est-à-dire la taille de la mémoire cache L2 : la pente suggère que cet cache accélère un peu les copies internes. Le creux se situe vers 16Ko, c'est-à-dire la taille de la mémoire cache L1. Entre ces deux extrémités, le temps utilisé par le noyau change du simple au double. Je me suis donc concentré sur la zone entre 1Ko et 64Ko sur la *figure 3b*. L'imprécision du chronométrage vient certainement de la manière dont le temps est mesuré, en *tics* du système au lieu de microsecondes ou autre. La durée d'exécution du programme étant comptée avec une très grosse louche, il y a beaucoup de variations d'une mesure à l'autre. De plus, Linux n'étant pas conçu pour être ultraprévisible (du moins pour la branche 2.4), les appels système peuvent suivre des chemins d'exécution différents si ça les arrange.

**NOTE**

Le noyau permet donc de lire un fichier assez efficacement avec une grande gamme de taille de blocs. Les facteurs limitant aux deux extrêmes sont :

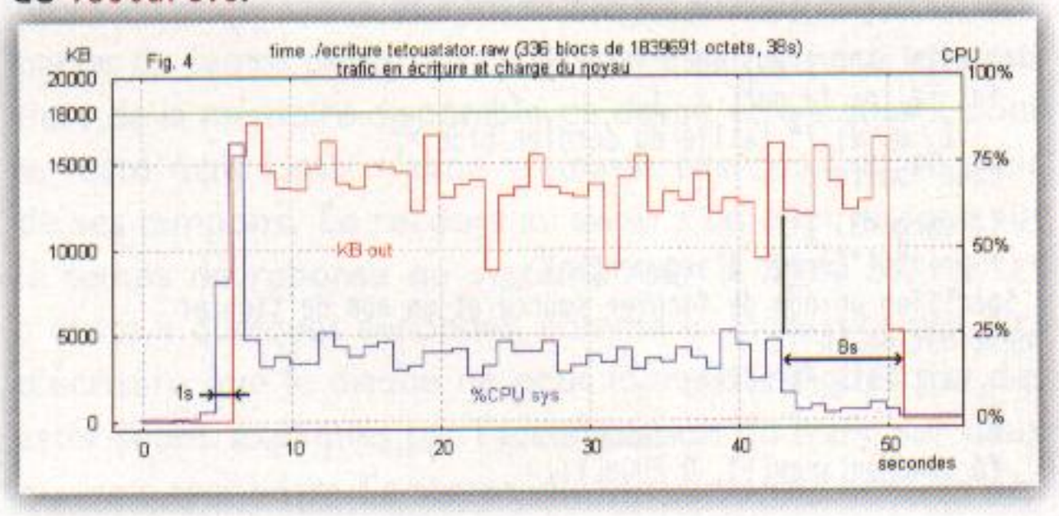
- Le temps de l'appel système, qui déclenche un changement de contexte relativement coûteux, pénalise les blocs très courts (< 4K0).
- La cache de lecture du noyau entraîne un doublement de la taille effective (l'empreinte en mémoire) d'un bloc. Si la taille d'un bloc est supérieure à la moitié de la mémoire totale, le noyau va devoir faire de la place pour la cache, ce qui entraîne la mise en *swap* du tampon en userland et du programme. Je vous laisse imaginer le bruit du crépitement du disque dur.

Des blocs de quelques dizaines de kilooctets représentent un bon compromis entre encombrement, latence, renouvellement de la mémoire cache du processeur... De plus, c'est proche de la taille des tampons de préchargement spéculatif (*prefetch*) du noyau et du disque dur (voir `man hdparm`).

## 2.4) Ecriture

L'objet de cette mesure est simplement de chronométrer le temps d'écriture pour la suite, puisque le comportement est probablement très proche de la lecture.

A ceci près que je désactive traditionnellement la cache d'écriture interne au disque dur, mais est-ce bien nécessaire sur un ordinateur portable, où les batteries prennent le relais en cas de coupure de courant ? De toute façon, la cache d'écriture du noyau prend le relais. La *figure 4* représente le débit, seconde par seconde, de l'écriture d'un fichier « vide » sur le disque, au moyen d'un petit programme ad hoc dérivé de `lecture.c`.




En voulant écrire un fichier de la taille exacte de `tetouatator.raw` pour obtenir des nombres aussi précis que possible, je me suis aperçu qu'on pouvait la factoriser pour utiliser des blocs de taille raisonnable, ce qui a conduit aux lignes suivantes :

```
>8 snip-snip >8
#define BUFFER_SIZE (1839691)
#define BUFFER_NB  (7*3*2*2*2*2)
>8 snip-snip >8
for (i=0; i<BUFFER_NB; i++) {
  t=write(fd_out, buffer, BUFFER_SIZE);
  if (t < BUFFER_SIZE) {
    perror("Erreur d'écriture");
    exit(EXIT_FAILURE);
  }
}
>8 snip-snip >8
```

Le temps d'exécution, variant entre 37,2s et 39s, est de 38,2s en moyenne. En raison de la grande taille du tampon et de la concision extrême du cœur de la boucle, le temps utilisé par le programme est tellement court que `time` indique 0. Le kernel occupe entre 6,7s et 7s, le processeur est donc exploité en moyenne à 18%.

Le débit est un peu moins rapide et le profil est plus saccadé que la lecture, puisqu'il faut en plus allouer de nouveaux *inodes*. On voit aussi que la cache d'écriture fonctionne à fond et on voit son effet :

- Au début, lorsque le programme la remplit à craquer (notez aussi le décalage d'une seconde) ;
- Et à la fin, lorsque l'écriture continue au moins 7 secondes après l'arrêt du programme (ce qui correspond bien à la taille de la RAM utilisable).

Le pic d'activité initial génère le même volume de données.

## 3) Effets combinés

Nos programmes filtres font appel à la lecture et à l'écriture presque simultanément. Mais il est physiquement impossible pour le disque dur d'effectuer ces deux opérations en même temps, en particulier parce que les adresses sont différentes (à moins de disposer de deux disques durs sur des interfaces différentes, ce qui n'est pas le cas le plus courant).

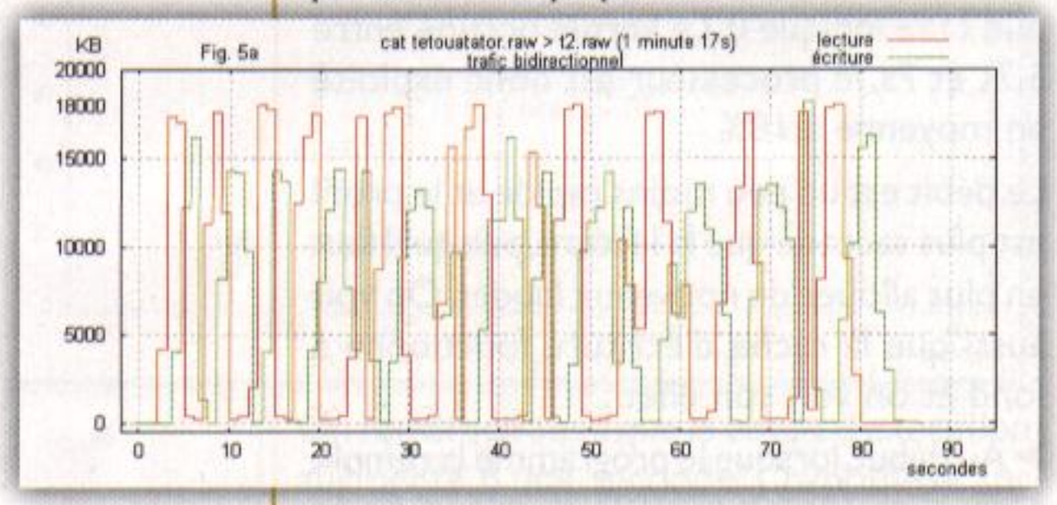
Dans les mesures effectuées ici, la partition contenant les données est relativement petite (10% de la taille totale du disque), ce qui réduit les mouvements de la tête et donc le temps perdu à la repositionner lors du passage du mode lecture au mode écriture et vice versa. Mais il faut bien alterner à un moment ou un autre.

Dans le cas présent, cette alternance est dictée par le système de fichier : `ext3` force une resynchronisation (*commit*) toutes les 5 secondes. La lecture va donc être interrompue par le noyau pour vider son cache d'écriture.

## 3.1) Alternance simple

La *figure 5a* trace le trafic en lecture et en écriture lors de la recopie du fichier en passant par le shell. Le temps total (indiqué par time, donc sans tenir compte du dernier commit mais ce n'est pas important) est de 77s. Normalement, 35s pour la lecture et 38s pour l'écriture donnent un total de 72s, il reste donc environ 5s perdues dans les « collisions » entre l'écriture et la lecture.

En regardant la *figure 5a*, on a l'impression que les accès sont un peu chaotiques. En toute logique, si on remet un peu d'ordre dedans, on devrait peut-être grappiller une partie du temps perdu.




Cette réflexion est donc suivie de la rédaction d'un autre programme qui va se servir d'un maximum de mémoire pour charger une partie du fichier, avant de le réécrire par gros morceaux. Le tout est de trouver la bonne taille du tampon pour ne pas entrer en conflit avec ceux du kernel. La meilleure taille pour lire les données ayant été déterminée autour de 16Ko, il faut effectuer de nombreuses opérations d'écriture et de lecture pour remplir un tampon beaucoup plus gros, supérieur à la moitié de la RAM. Le tout est de trouver comment empêcher le kernel de mettre le programme en swap pour libérer de la mémoire pour son propre cache (désactiver la swap n'étant pas une solution satisfaisante). Autre question : est-il nécessaire de « chauffer la mémoire » ? Sachant que Linux a une allocation « optimiste » (selon `man malloc`), la mémoire allouée ne sera même pas prête à l'utilisation au retour de `malloc()`. C'est la première référence (écriture ou lecture) sur *chaque* page qui va déclencher la traduction d'adresse. L'idée serait d'effectuer un premier accès à chaque page du bloc alloué pour forcer la translation avant la lecture. Mais il y a deux facteurs limitant :

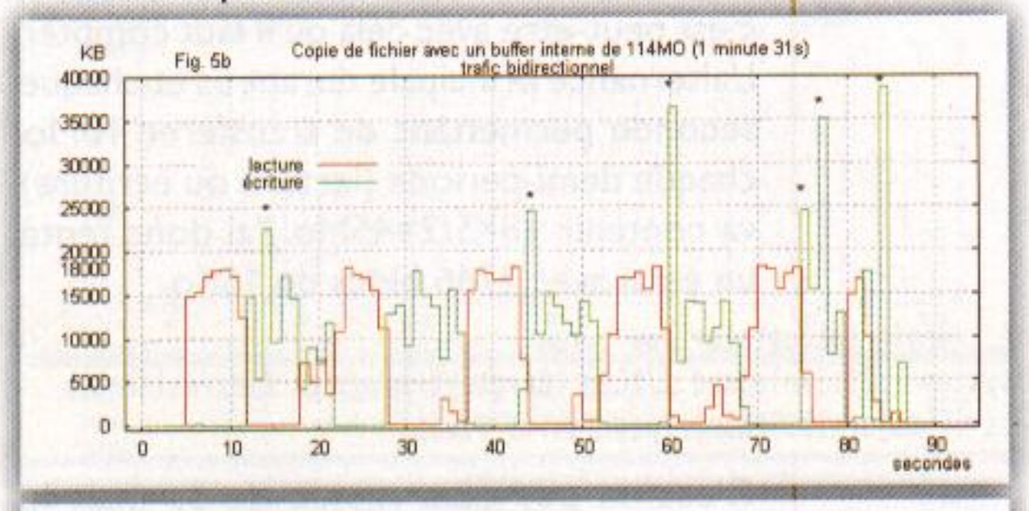
- D'abord, sur x86, les pages font 4Ko (le support des grandes pages n'étant pas disponible partout), il y a un nombre limité d'entrées dans la TLB et elle ne peut pas contenir tout le *mapping* de la RAM, ni de notre immense tampon. Le temps d'accéder à la fin du tampon, la TLB aura été remplacée plusieurs fois.
- La lecture est assez lente pour permettre aux autres opérations internes de s'exécuter « en même temps » (heureusement).

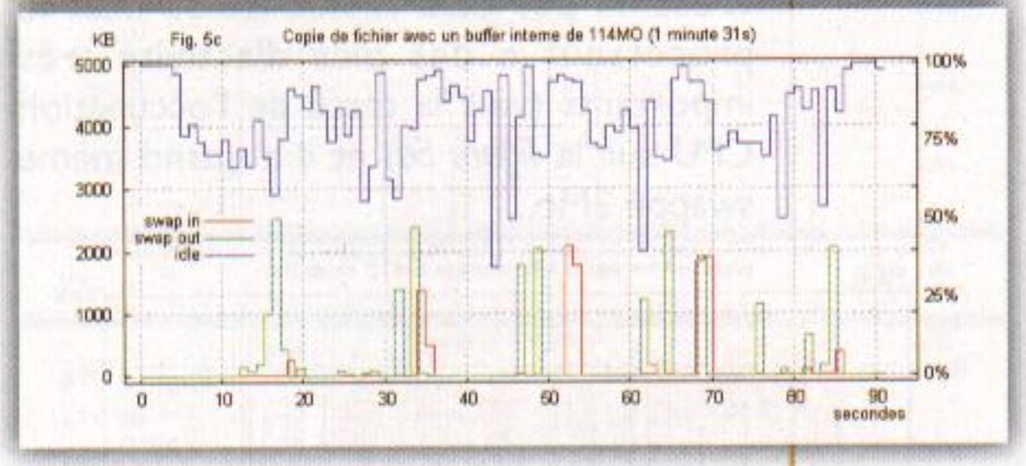
Sans oublier qu'une procédure de chauffe serait une étape « sérialisante » pour le programme, c'est-à-dire un retard inutile pour passer à la lecture proprement dite. Par contre, il y a une autre solution intermédiaire : utiliser le temps *idle* pour chauffer les prochains blocs. Mais on ne peut pas encore utiliser ce temps car `read()` et `write()` sont bloquants et ne redonnent la main au programme que lorsqu'ils ont terminé leur travail. Il faudrait du code *multithread* ou bien faire appel à `select()`. Et encore, il reste à prouver que cette méthode permet réellement de gagner du temps. De plus, le temps « libre » ne sera plus disponible lorsque le filtre remplira sa fonction, au moyen de code qui prendra probablement plus de temps à exécuter que `read()` et `write()` réunis. Nous voilà donc avec `copie.c`, un programme encore simple, constitué essentiellement d'une paire de boucles.

```
/* fichier copie.c
créé par Yann GUIDON (whygee@f-cpu.org)
version Mon Aug  1 13:14:06 CEST 2005

 fonction :
   copie un fichier le plus vite possible
   au moyen d'un gigantesque buffer !

 compilation :
   gcc -Wall -g -o copie copie.c

 invocation :
   copie nom_src nom_dest
*/

#define _FILE_OFFSET_BITS 64
#define _LARGEFILE_SOURCE
#define _LARGEFILE64_SOURCE

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>  /* open() */
#include <errno.h>  /* perror() */
#include <stdio.h>  /* fprintf(), fflush() */
#include <stdlib.h> /* malloc() */
#include <unistd.h> /* exit(), read() */

#ifndef BLOCK_SIZE
#define BLOCK_SIZE (16*1024)
#endif

/* 7000*16K=114MB */
#ifndef BLOCK_NUMBER
#define BLOCK_NUMBER (7000)
#endif

#define BUFFER_SIZE (BLOCK_SIZE*BLOCK_NUMBER)

int main (int argc, char *argv[]) {
  int fd_in, fd_out, i, j,
     t, u, v; /* taille du dernier bloc */
  char *buffer;

  if (argc<3) {
    printf("Erreur d'argument\n\
 Spécifier un nom de fichier source et un nom de fichier
destination.\n");
    exit(EXIT_FAILURE);
  }

  fd_in=open(argv[1], O_RDONLY);
```

```
if (fd_in==-1) {
  perror("Erreur à l'ouverture de la source");
  exit(EXIT_FAILURE);
}
fd_out=open(argv[2], O_WRONLY|O_CREAT);
if (fd_out==-1) {
  perror("Erreur à l'ouverture de la destination");
  exit(EXIT_FAILURE);
}
buffer=malloc(BUFFER_SIZE);
if (buffer==NULL) {
  perror("Erreur de malloc() ");
  exit(EXIT_FAILURE);
}
while(1) {
  while(1) {
/* remplissage du buffer */
    i=0;
    do {
      t=read(fd_in, buffer+(i*BLOCK_SIZE), BLOCK_SIZE);
      if (t<0) {
        perror("Erreur de lecture");
        exit(EXIT_FAILURE);
      }
    } while ((t==BLOCK_SIZE)
        && (++i<BLOCK_NUMBER));
        /* i n'est incrémenté que si (t==BLOCK_SIZE) */
/* vidange */
    j=0;
    do {
      /* détermine la taille du bloc à écrire */
      if (j==i) { /* fin de la boucle interne */
        if (t==0)
          exit(EXIT_SUCCESS);
        else {
          u=t;
          t=BLOCK_SIZE; /* force la sortie de cette boucle */
/* t=0 serait plus radical (entrainant
 la sortie directe du programme à la prochaine itération)
 mais man read dit que ce n'est pas nécessairement
 la fin d'un fichier si read retourne un bloc plus petit que
demandé. */
        }
      }
      else
        u=BLOCK_SIZE;
      v=write(fd_out, buffer+(j*BLOCK_SIZE), u);
      if (u!=v) {
        perror("Erreur d'écriture");
        exit(EXIT_FAILURE);
      }
      j++;
    }
    while ((j<i) || (t!=BLOCK_SIZE));
     /* engendre un "off-by-one" s'il reste un morceau */
  }
}
```

Avec un tel programme, on se dit que ça devrait aller. En fait, pas du tout, c'est même la catastrophe. Le temps total d'exécution est 1min.31s, le résultat est donc défavorable. Mais surtout, la *figure 5c* montre que le système déplace des blocs en provenance et vers le swap ! Ce qui explique une partie du temps perdu. Le programme alloue juste les deux tiers de la mémoire disponible, ce devait être suffisant pour le reste mais Linux n'aime vraiment pas que l'on se mêle de ses tampons. Le recours au swap a un effet marqué sur le temps de réponse du système : sur la *figure 5b*, `vmstat` « saute » quelques secondes, comme le montrent des pics d'écriture que le disque ne peut fournir (indiqués par des astérisques, expliqués par l'accumulation du trafic sur deux ou trois secondes). Le temps *idle* montre aussi des pics peu encourageants. Ce n'est donc par la bonne direction à prendre.



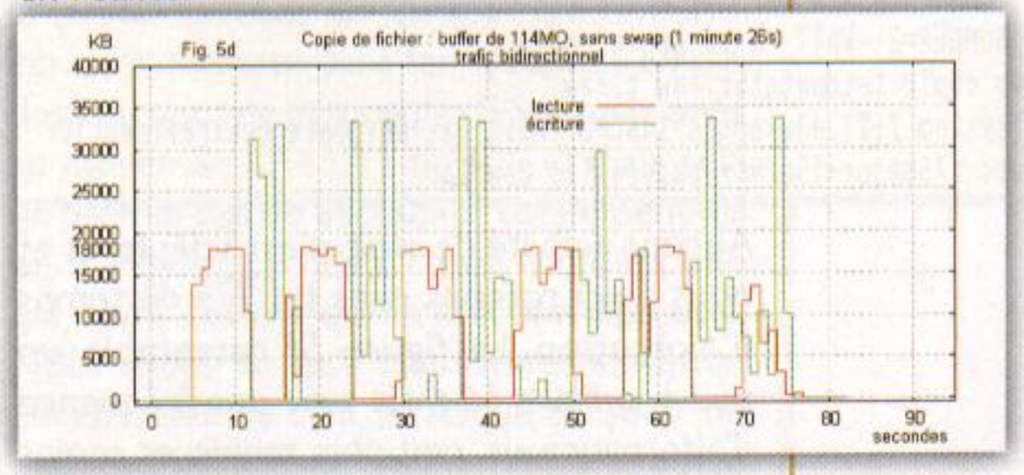




## 3.3) Je suis root chez moi, après tout !

Comment empêcher Linux de swapper ? J'ai tenté de lui enlever son swap chéri pour voir.

```
$ swapoff -a
```

et le temps écoulé descend un peu à 1:26.91s. Pourtant ça ne l'empêche pas de « perdre les pédales » comme le montrent encore les pics de la *figure 5d*, impossibles à atteindre en réalité.




Ce n'est donc pas la bonne voie, alors

```
$ swapon -a
```

## 3.4) Un peu plus en douceur ?

Réduction du tampon à 64Mo, moins de la moitié de la mémoire installée. Le temps d'exécution total reste à 1min. 26s. Le kernel swappe toujours un petit peu mais `vmstat` n'a apparemment plus le hoquet. Linux semble continuer à lire le disque « au compte-gouttes » pendant qu'il écrit, ce qui ralentit un peu l'ensemble.

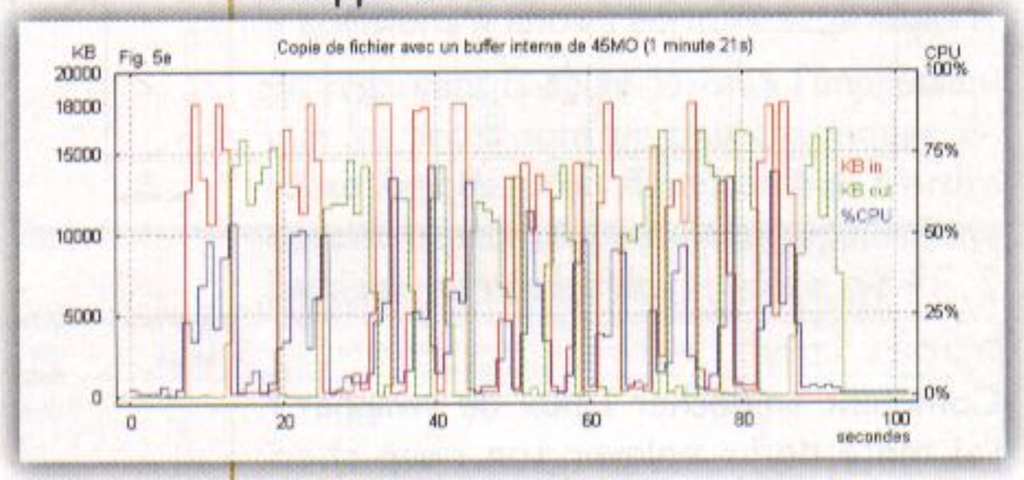
## 3.5) Il faut lui dire avec des fleurs, alors ?

Je me suis peut-être trompé sur la variable à synchroniser avec le noyau récalcitrant.

Puisque Linux semble décidé à vouloir faire papillonner la tête du disque dur, c'est peut-être avec cela qu'il faut compter. L'alternance principale durant 5s et chaque seconde permettant de transférer 18Mo, chaque demi-période (lecture ou écriture) va contenir 18×5/2=45Mo. J'ai donc tenté un essai avec 2746 blocs de 16Ko.

```
$ /usr/bin/time ./copie tetouatator.raw t.raw
0.06user 14.51system 1:21.77elapsed 17%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (75major+10992minor)pagefaults 0swaps
```

C'est un peu plus rapide de 5s mais le processeur a des pics d'activité très importants (voir la tracé de l'occupation CPU sur la *figure 5e*) et il a quand même swappé 2Mo...



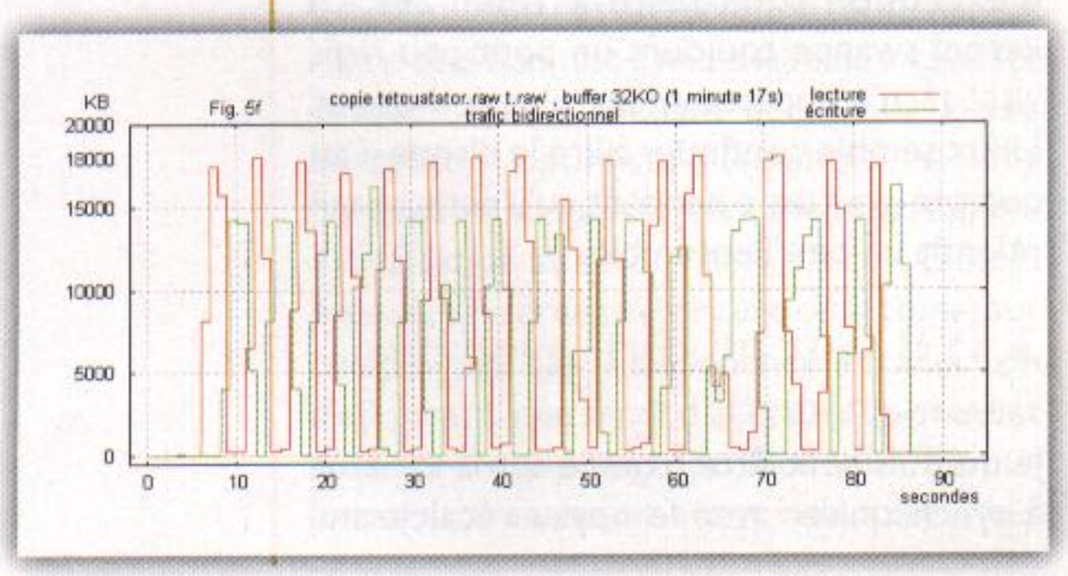

## 3.6) Retour à la case départ

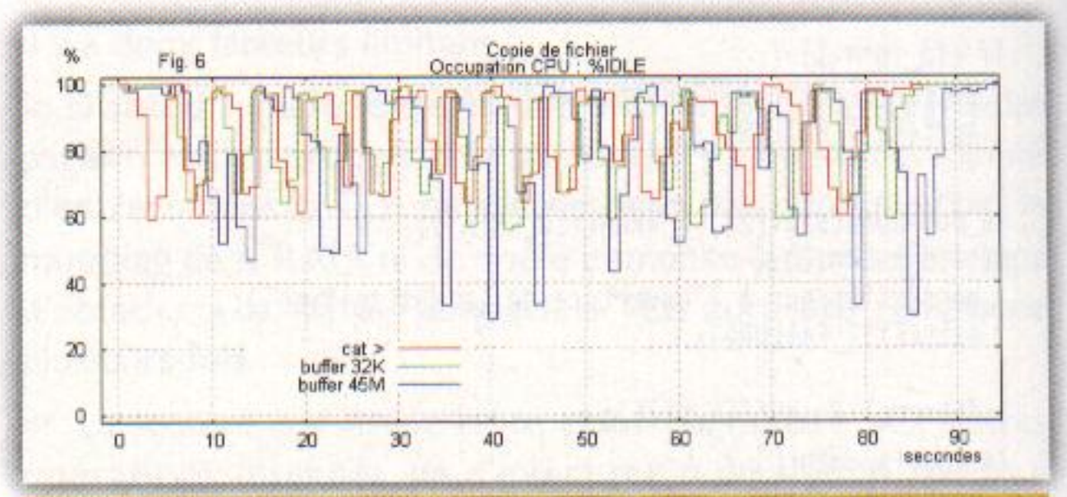
Il semble bien que l'impression de départ soit mauvaise, puisque le temps d'exécution diminue avec la taille du tampon. Un dernier essai s'impose donc avec une taille beaucoup plus réduite, 2 blocs de 16Ko feront l'affaire.

```
$ gcc -DBLOCK_NUMBER=2 -Wall -g -o copie copie.c
$ /usr/bin/time copie tetouatator.raw t.raw
0.09user 11.07system 1:17.43elapsed 14%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (76major+15minor)pagefaults 0swaps
```

Aucune activité de swap n'est à déplorer et nous voici revenus dans les 77s de temps d'exécution. La *figure 5f* ressemble un peu à la *figure 5a* mais avec une fréquence d'alternance un peu plus rapide et moins désordonnée.

La *figure 6* compare le temps CPU disponible selon le type d'approche. L'utilisation d'un grand buffer, en plus d'allonger l'exécution, entraîne des pics d'activité qui peuvent être préjudiciables pour certaines applications.




**NOTE**

Le noyau Linux s'occupe tout seul de déterminer la taille des transferts avec le disque dur, il est donc inutile d'utiliser un grand tampon de copie. Linux alloue un maximum de place pour ces derniers, il vaut donc mieux garder la taille de nos programmes aussi petite que possible si on ne veut pas que le système se mette à swapper.

## 4) Influence de la charge

Un troisième élément, et non des moindres, vient alors s'ajouter : il faut bien que notre programme-filtre remplisse une fonction. Pour l'instant, pas besoin d'une utilité particulière, il faut juste voir comment le système se comporte quand on ajoute des instructions de traitement.

## 4.1) Un filtre simple

La « charge » doit répondre à quelques contraintes : elle doit être facilement et finement réglable, très facile à réaliser, proche d'une utilisation réelle et solliciter uniquement le processeur pour ne pas faire intervenir d'autres paramètres. En plus, le compilateur ne doit pas pouvoir l'« optimiser » pour éviter que le programme n'ait aucun effet. La solution retenue est d'effectuer un XOR sur chaque octet du buffer, ce qui fait en plus travailler le système de mémoire cache intégré. Le programme `filtre_simple.c` est une modification du source précédent, à la différence que le corps de la boucle principale a la forme suivante :

```
  while ( (t = read(fd_in, buffer, BUFFER_SIZE)) > 0 ) {
/* ne fait absolument rien mais grille du temps CPU */
    for (j=0; j<charge; j++)
      for (i=0; i<t; i++)
        buffer[i]^=j;
    write(fd_out, buffer, t);
  }
```

Chaque fois que la boucle interne est exécutée, tout le buffer d'entrée est balayé et chaque octet est légèrement modifié. Le XOR est effectué avec le compteur de boucle externe pour empêcher qu'un compilateur trop malin ne simplifie le code. En fait, ce n'est pas le XOR qui prend du temps, mais l'accès à la mémoire et le découpage en octets (cela nécessite des conversions internes sur les processeurs PII et supérieurs). La « charge » est le nombre de passes et on peut la définir au démarrage du programme pour éviter de le recompiler à chaque fois. On peut remarquer que si la charge est une puissance de deux (et supérieure à deux), la sortie est identique à l'entrée (exercice pour les débutants en programmation : expliquez pourquoi). Pour commencer, un premier essai est effectué avec une charge de 4. Le temps d'exécution moyen est de 113s ou plus exactement :

```
$ rm -f t.raw && sync && /usr/bin/time -f '%e %U %S %P \
 ./filtre_simple tetouatator.raw t.raw
```

résultats de 5 exécutions consécutives :

```
total  user   sys  CPU
113.98 59.46 10.90 61%
111.99 58.33 11.42 62%
113.57 59.44 9.89  61%
113.12 58.97 10.68 61%
113.34 59.24 10.25 61%
```

Si le temps *user* est de presque 60s pour 4 passes, alors chaque passe occupe environ 15s de temps CPU (soit, rapporté au fichier entier, environ 41 millions de XOR par seconde). On pourrait affiner la durée de chaque passe en sautant des octets, mais la bande passante de la mémoire cache atteindrait sa limite. La *figure 7a* montre les débits en lecture et en écriture. Pour une fois, la lecture est plus « lente » que l'écriture, ce qui est un premier signe qu'il se passe quelque chose de différent. Les alternances sont bien régulières avec une période de 5s. L'écriture atteint des pics de 16Mo/s, signifiant que le noyau fonctionne à fond.

Mais le plus intéressant est sur la *figure 7b*, qui montre un détail de l'activité en lecture et en écriture. Essentiellement, le processeur est au repos lors de l'écriture et hyperactif lors de la lecture, où le programme occupe environ 60% de la machine et le noyau 40%.

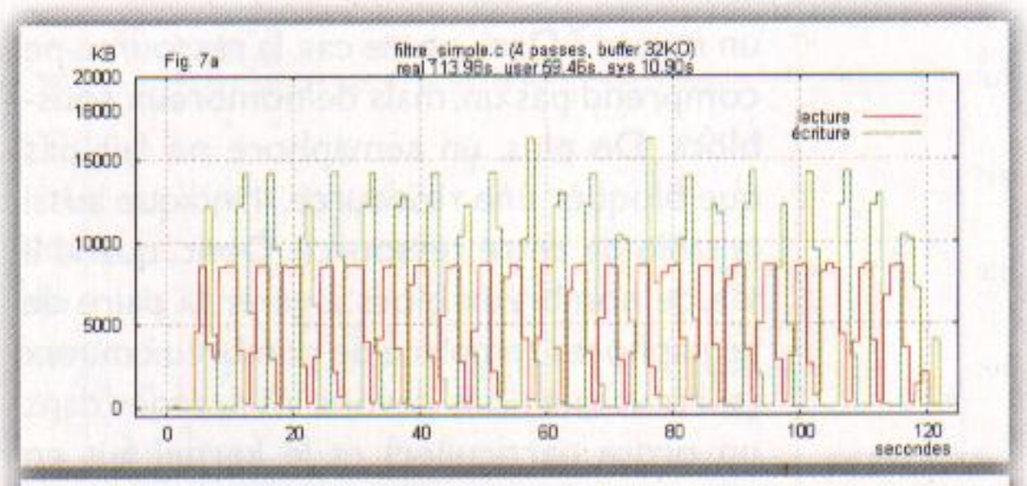
La raison est simple : le kernel retourne les données de lecture immédiatement au programme mais il attend de remplir son buffer ou qu'un commit automatique se déclenche pour écrire. Le programme utilisateur, en particulier sa charge, est donc dépendant de la lecture et n'effectue rien pendant que le noyau vide son cache d'écriture. Voici un nouvel axe d'amélioration !

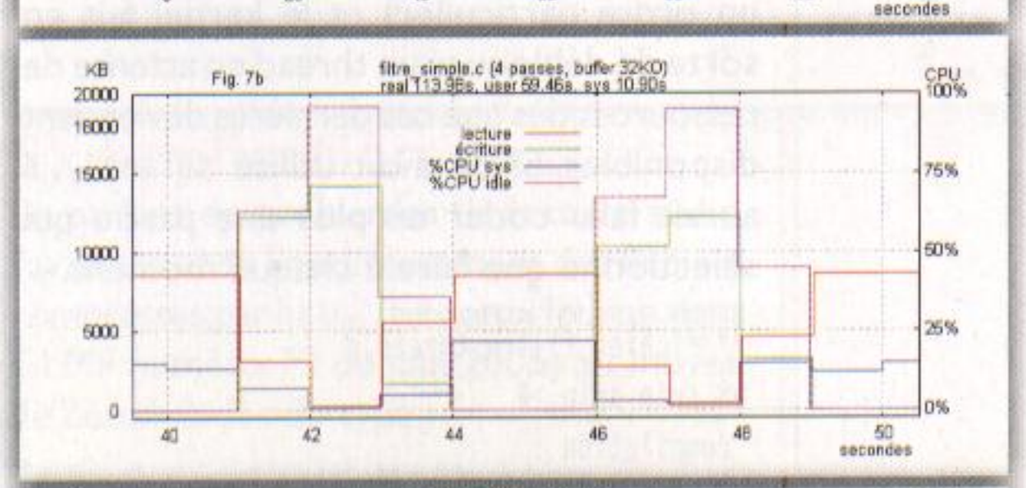
La *figure 7c* synthétise ce comportement avec, d'une part, les temps totaux et *user* et, d'autre part, l'occupation du CPU.

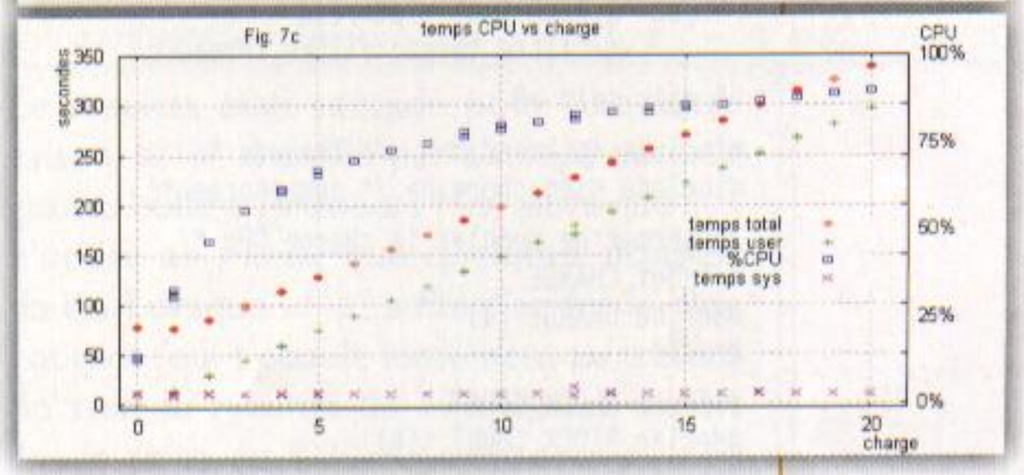
- La charge est totalement linéaire et le temps *user* est affiché par une belle droite qui part de zéro.
- Le temps système est constant (à environ 10s).
- Le temps total dépend de la somme du temps système et du temps *user*, avec un minimum défini par le temps d'accès au disque (mesuré auparavant à 77s). Le tracé du temps total est donc défini par deux droites (une constante et une linéaire).
- La différence entre le temps total et la somme système+*user* est le temps d'écriture, mesuré à environ 40s dans la deuxième partie (ce qui correspond à l'écartement entre les deux droites).
- Le CPU n'est donc jamais totalement occupé comme le montre la belle courbe en `C - 1/x`. L'occupation du processeur ne sera jamais totale à cause des 40 secondes d'écriture.

Au passage, ce graphe valide aussi le principe du système de charge et nous donne une première mesure des caractéristiques du système.

Normalement, le processeur devrait être totalement occupé (*idle* à 0%) avec une charge à 4 pour cet ordinateur (cela varie d'un système à l'autre).




## 4.2) Même charge mais programme différent

La solution est claire : il faut « découpler » la partie lecture du couple traitement-écriture. Il existe dans les systèmes UNIX des fonctions de gestion des E/S asynchrones au moyen de `select()` ou `poll()` mais cela ne permet pas de découpler totalement les deux parties, car il y en aura toujours une qui devra décider quelle fonction exécuter. Nous avons vu qu'il faut laisser le kernel décider seul de cela, au risque de subir de lourdes pénalités. En fait, mon choix se porte directement sur le modèle « *multithread* » en raison de précédentes expériences « quasi-temps réel » et de la disponibilité des fonctions POSIX. Dans notre cas, cela permet de simplifier et raccourcir le code tout en le gardant portable et parallélisable dans le futur.

Le fichier `filtre_thread.c`, dont les lignes importantes sont données ci-dessous, utilise une paire de *sémaphores* pour gérer la ressource du tampon de mémoire global. Dans le cas le plus simple (avec juste un seul bloc de 16Ko), les sémaphores implémentent un protocole de « poignée de main » (*handshaking*) qui empêche un des *threads* d'utiliser l'unique bloc pendant que l'autre l'accède. Pourquoi alors ne pas utiliser

un *mutex* ? Dans notre cas, la ressource ne comprend pas un, mais de nombreux sous-blocs. De plus, un sémaphore ne fait pas que bloquer une ressource, il indique aussi la taille de cette ressource. Donc, quand il y a de nombreux blocs à gérer, la paire de sémaphores remplace de nombreux mutex (surtout que nous devons les accéder dans un ordre particulier) et le kernel fait en sorte de débloquer un thread en attente de ressources dès que ces dernières deviennent disponibles. Si on avait utilisé `select()`, il aurait fallu coder en plus une partie qui sélectionne que faire à chaque moment.

```
/* fichier filtre_thread.c
>8 snip-snip >8
 compilation :
   gcc -D_REENTRANT -lpthread \
       -o filtre_thread filtre_thread.c
>8 snip-snip >8
#include <pthread.h>   /* threads */
#include <semaphore.h> /* semaphores */
/* permet de moduler la charge CPU */
#ifndef CHARGE
#define CHARGE (4)
#endif
#ifndef BLOCK_COUNT
#define BLOCK_COUNT (16)
#endif
#ifndef BLOCK_SIZE
#define BLOCK_SIZE (16*1024)
#endif
#define BUFFER_SIZE (BLOCK_COUNT*BLOCK_SIZE)
char *buffer;
int charge = CHARGE, fd_in, fd_out;
ssize_t t = 0;
ssize_t byte_count[BLOCK_COUNT];
sem_t sem_lu, sem_ecrit;
pthread_t thr_lecture;
/* Cette partie du code se charge de remplir
*buffer avec des
  blocs, tant qu'il y reste de la place et des
octets à lire : */
void * thread_lecture (void * arg) {
  ssize_t t;
  int index=0, index_block=0;
  do {
    sem_wait(&sem_ecrit);
    t = read(fd_in, &buffer[index_block],
BLOCK_SIZE);
    byte_count[index] = t;
    if (t < 0)
      perror("Erreur de lecture");
    else {
      /* renvoie le résultat et incrémente les
compteurs locaux */
      index_block += BLOCK_SIZE;
      if (++index >= BLOCK_COUNT)
        index = index_block = 0;
    }
    sem_post(&sem_lu);
  } while (t > 0);

  /* mort naturelle sans conséquence : */
  return NULL;
}
/* En plus d'effectuer toute l'initialisation,
le main() contient l'autre
 partie du programme qui effectue le "vrai
travail" et écrit le résultat : */
int main (int argc, char *argv[]) {
  unsigned long int i;
  int index=0, index_block=0, j;
>8 snip-snip >8
  /* initialisation des sémaphores */
  sem_init(&sem_ecrit, 0, BLOCK_COUNT);
  sem_init(&sem_lu, 0, 0);
  /* création du thread de lecture */
  if (pthread_create (&thr_lecture, NULL, thread_lecture, NULL)) {
    perror("Erreur à la création du thread de lecture");
    exit(EXIT_FAILURE);
  }
  /* boucle d'attente-xoriture-écriture */
  while(1) {
    sem_wait(&sem_lu);
    t = byte_count[index];
    if (t > 0) {
      /* charge le CPU */
      for (j = 0; j < charge; j++)
        for (i = 0; i < t; i++)
          buffer[i+index_block] ^= j;
      write(fd_out, &buffer[index_block], t);
      index_block += BLOCK_SIZE;
      if (++index >= BLOCK_COUNT)
        index = index_block = 0;
      sem_post(&sem_ecrit);
    }
    else {
      pthread_join(thr_lecture, NULL);
      exit(EXIT_SUCCESS);
    }
  }
}
```

Avec le système des doubles sémaphores, le codage est relativement simple (même si l'expérience aide toujours et je vous invite à lire les pages `man` de toutes les fonctions nouvelles pour vous). La paire fonctionne un peu comme des vases communicants et si on regarde la succession des valeurs des sémaphores, on s'aperçoit que leur somme est toujours égale (à une unité près) au nombre total de blocs dans le tampon. Après un peu de débuggage pour enlever les quelques bêtises de syntaxe qui m'étaient passées sous le nez, je tente un premier essai :

```
$ gcc -D_REENTRANT -lpthread -o filtre_thread filtre_thread.c
$ /usr/bin/time -f '%e %U %S %P' ./filtre_thread tetouatator.raw t.raw
129.99 72.48 14.05 66%
```

Aucune activité de swap n'est détectée, mais le processeur reste inactif durant les périodes d'écriture. Pire : le temps total d'exécution est bien supérieur aux 113s mesurées auparavant.

## 4.3) Le retour du gros tampon

En fait, la taille du tampon apparaît bien trop petite. Sur les informations retournées par `vmstat` sur la *figure 8a*, on voit que le kernel écrit des quantités de 20 à 25Mo à chaque fois et le petit tampon de 256Ko ne suffit pas du tout. J'augmente donc le nombre de blocs et le résultat est net :

```
95.26 72.14 15.22 91%
```

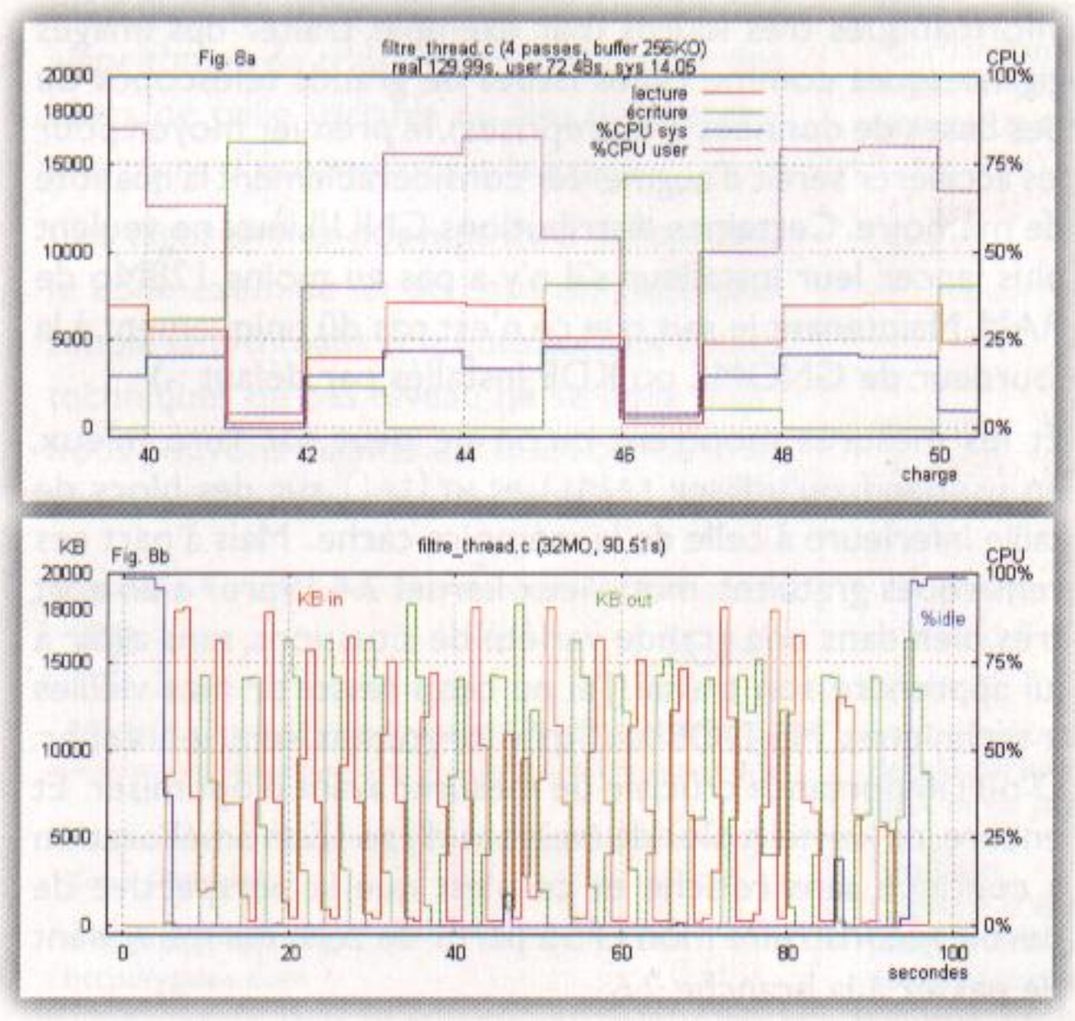
Les 113s sont oubliées ! Mais c'est encore loin des 77s chronométrés sans charge et les rafales d'écriture bloquent encore sporadiquement l'exécution du programme. Il faut encore augmenter le nombre de blocs et le prochain essai en compte 2048, soit 32Mo au total :

```
90.51 71.37 16.34 96%
```

On a encore gagné 5s mais on ne peut pas faire mieux, car il n'y a presque plus de temps CPU libre. De plus, une légère modification du code de charge a un peu augmenté le temps d'exécution de chaque passe. Cela pourrait changer en jouant sur les directives d'optimisation du compilateur, mais le programme est maintenant *CPU bound* : le total des 71s de traitement plus les 10s du système est plus long que 77s sans charge. La *figure 8b* montre d'ailleurs que le temps *idle* est presque tout le temps nul. Grâce au script suivant, on peut tracer le profil comparatif de l'activité, qui est présenté sur la *figure 8c*.

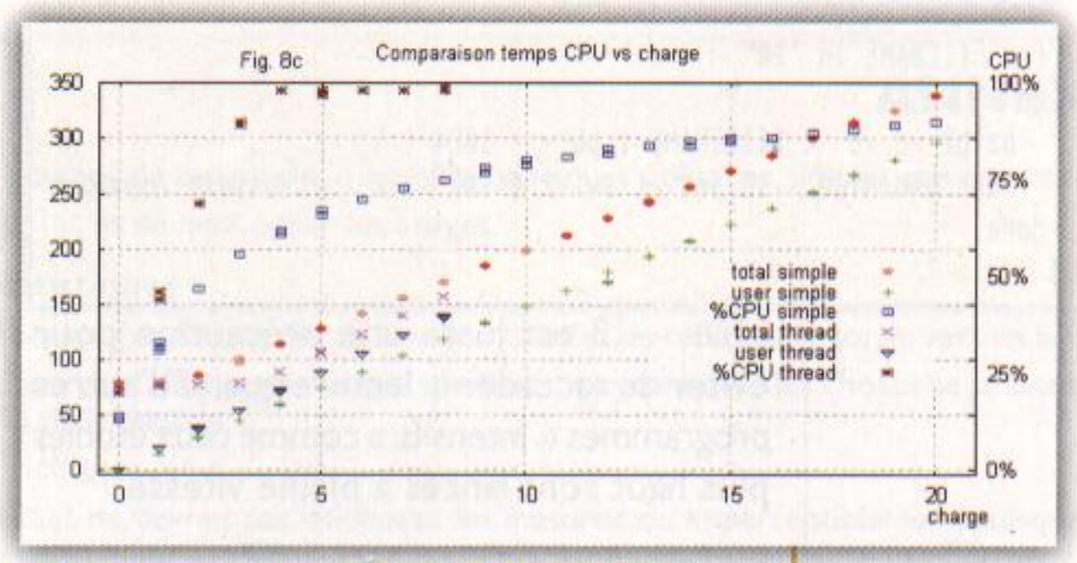
```
for i in $(seq 8 -1 0)
do
  echo $i:
  for j in 1 2 3
  do
    rm -f /common/t.raw
    sync
    echo $i $(/usr/bin/time -f '%e %U %S %P' \
      ./filtre_thread tetouatator.raw t.raw $i 2>&1 ) >> thread.out
  done
done
```

On observe la légère différence de pente pour le temps *user* due à une petite modification du code (une simple histoire de mode d'adressage non optimisé). Mais on voit surtout que la pente de l'occupation du CPU est beaucoup plus raide et l'augmentation du temps total se produit sous une charge plus élevée. L'amélioration est donc significative pour des charges moyennes, telles que le traitement ou la compression/décompression de son ou d'images. L'effet est moins marqué pour les charges très fortes (bien qu'un gain de quelques dizaines de secondes soit toujours bienvenu) et la limitation de la bande passante du disque dur rend l'utilisation de threads trop lourde s'il faut faire moins de deux passes. Mais ces conclusions ne sont pas définitives car elles dépendent de l'ordinateur qui fait tourner le programme. Une accélération du processeur ou du disque dur changerait le bilan. Une chose est sûre : sans recourir à `nice`, la lecture d'un fichier MP3 simultanément à l'exécution du programme multithreadé donne une écoute très saccadée.










## 5) Application directe

En parlant de problèmes de lecture audio, j'avais tenté de lire directement des fichiers compressés par `bzip2` (tels ceux fournis dans GLMF numéro 73 de juin 2005) au moyen de commandes du type :

```
bzip2 -d -c -k fichier.raw.bz2 | \
  /usr/bin/nice --10 sox -t sw -r 44100 -c 2 - -t ossdsp /dev/dsp
```

Le résultat était saccadé pour une autre raison. `bzip2` travaille par à-coups avec de grands buffers (plusieurs Mo) alors que `sox` n'utilise au mieux que quelques dizaines de Ko. Lorsque `bzip2` a fini d'écrire le bloc courant (qui s'écoule lentement au travers de sox, au rythme de 176,4Ko/s), il doit lire la suite du fichier compressé puis le décompresser, ce qui prend beaucoup de temps. Le petit tampon de `sox` et du kernel a déjà expiré et la lecture est momentanément interrompue...

En modifiant le programme multithreadé précédent, on résout facilement le problème. Pour cela, il suffit de faire quelques petits aménagements :

- Permettre de lire et écrire sur `stdin`/`stdout`. C'est l'affaire de quelques lignes supplémentaires. Comme pour `sox`, j'ai choisi « - » pour signifier qu'il s'agit de l'entrée/sortie standard.
- Enlever le code de charge fictive.
- Permettre de spécifier la taille totale du tampon sur la ligne de commande. Il faut faire quelques petits aménagements car toutes les tailles étaient en constantes.
- Ne pas oublier de fermer les fichiers en sortant ! Sinon `sox` reste figé alors que le programme est déjà terminé et sorti de la mémoire.

Le résultat s'appelle `pb.c` pour « *Pipe Buffer* ». Les clics ont tous disparu (alors qu'il y en avait une douzaine par minute). J'en ai profité pour faire un joli script qui est logé dans `/usr/bin/playbz2` :

```
#!/bin/bash
if [ -z "$1" ]
then
  echo "Missing argument : file name"
else
```

```
  for FILENAME in "$@"
  do
    bzip2 -d -c -k $FILENAME | pb - - 1024 | \
    /usr/bin/nice --10 sox -t sw -r 44100 -c 2 - -t ossdsp /dev/dsp
  done
fi
```

`nice --10` est juste une précaution pour éviter de saccader la lecture quand d'autres programmes « intensifs » comme ceux étudiés plus haut sont lancés à pleine vitesse.

## 6) Améliorations possibles

- J'ai choisi de découper le programme en deux, car il semblait que la sérialisation de la lecture avec le traitement était le frein central. Il se peut que des systèmes éprouvent aussi des difficultés en écriture, par exemple si la mémoire est très petite ou si le noyau est trop primitif. Dans ce cas, il suffit de créer un autre thread pour l'écriture (et croiser les doigts pour qu'une forme de threads soit disponible si on ne veut pas travailler directement au niveau des interruptions).
- La taille du tampon a été déterminée expérimentalement pour une plate-forme particulière et ne conviendrait probablement pas si le rapport des vitesses entre le disque et le processeur change. Au lieu de le déterminer à la main pour chaque machine, il faudrait créer un algorithme *adaptatif* qui augmenterait la taille du tampon au besoin. Cet algorithme est trop compliqué pour le présent article. De plus, la plupart des applications qui utiliseront le code multithreadé seront adaptées au cas par cas et probablement même pour une machine particulière et des charges spécifiques.
- Nous avons étudié des cas où l'information ne subissait pas de modification de bande passante. Dans le cas d'un compresseur ou d'un décompresseur, il faut tenir compte du fait que la quantité de données lues n'est pas la même qu'en écriture. Le système de sémaphores et de gestion des blocs devrait encore fonctionner mais une vérification formelle s'impose.

## 7) Conclusion

### Méthodologie

Cet article a présenté un cas simple d'« optimalisation » : commencer par examiner les opérations les plus simples et les plus rapides puis ajouter progressivement des éléments en essayant de quantifier et réduire leur impact sur les performances. Optimiser un programme revient classiquement à le regarder et remplacer des blocs jugés inefficaces par d'autres que l'on estime meilleurs. Par contre, reconstruire un programme en partant de sa fonction la plus fondamentale est une stratégie d'optimisation qui empêche de se faire piéger dans des conjectures (qui sont souvent fausses et ralentissent le programme au lieu de l'accélérer). Bien sûr, il ne faut jamais oublier de chronométrer le résultat pour vérifier que les modifications ont eu un effet et qu'il est positif. Nous avons vu que c'est aussi très instructif !

### Linux

L'utilisation de buffers noyau pour `read()` et `write()` peut devenir une catastrophe à cause de la copie entre l'espace noyau et l'espace *user*. Ce serait acceptable, s'il ne fallait pas compter avec son envie irrépressible de swapper tout ce qu'il peut pour faire de la place pour ses tampons en mémoire, parfois inutilement immenses. Et ce n'est que la partie émergée du problème :

- La compatibilité `POSIX` et la base logicielle existante ne permettent pas d'éviter cette duplication qui, dans certains cas, n'a aucun sens : on ne peut pas faire un `read()` de plus de la moitié de la mémoire vive, au risque de faire swapper nos propres données *user* pour libérer de la place pour le buffer du kernel ! Et même avec un simple tampon utilisant seulement le quart de la mémoire, il trouve encore le moyen de swapper quelques morceaux (c'est à se demander s'il faut vraiment activer le swap).
- L'option `READ_O_DIRECT` ne fonctionne pas sur les fichiers normaux, du moins sur ma version de kernel. Affaire à suivre de près !
- `mmap()` est un outil indispensable, mais en userland, c'est « la cause de tous les maux » (par rapport à cela, on peut mettre des `goto` partout les yeux fermés). En plus d'être synchrone, sérialisant, et surtout avec une latence imprévisible, il faut l'utiliser avec une prudence extrême et prévoir beaucoup de code pour éviter de faire des bêtises : le mieux est l'ennemi du bien ! (Ce sont encore ces réminiscences du *benchmarking* de BLAST qui frappent encore, des zigotos n'avaient pas imaginé qu'il faudrait utiliser des bibliothèques de protéines de plus de 1,8GO sur une plate-forme Linux/ia32)

Je ne vais pas paraître très original, mais vu le comportement du kernel considéré ici, s'il fallait faire des traitements informatiques très lourds (par exemple, traiter des images gigantesques comme celles issues de grands téléscopes ou des bases de données d'entreprises), le premier moyen pour les accélérer serait d'augmenter considérablement la quantité de mémoire. Certaines distributions GNU/Linux ne veulent plus lancer leur installeur s'il n'y a pas au moins 128Mo de RAM. Maintenant je sais que ce n'est pas dû uniquement à la lourdeur de GNOME ou KDE installés par défaut ;-)

Et les mesures montrent qu'on ne peut pas faire mieux, en userland, qu'utiliser `read()` et `write()` sur des blocs de taille inférieure à celle de la mémoire cache. Mais à part ces remarques gratuites, mon vieux kernel 2.4.19pre7 s'en sort très bien dans une grande variété de situations, sans avoir à lui apprendre son travail. J'ai eu beau ressortir mes vieilles « techniques MS-DOS », il m'a longtemps tenu en échec. D'où l'importance cruciale de mesurer avant d'optimiser. Et encore, ce kernel a plus de trois ans d'âge ! Son amélioration a continué sans relâche et ce n'est que la perspective de devoir reconstruire mon LFS à partir de zéro qui me retient de passer à la branche 2.6.

## Parallélisme :

**Autre chose à retenir :**

Un programme qui utilise `select()` ou `poll()` n'est pas parallélisable alors qu'en utilisant des threads, :

- On laisse le kernel travailler proprement (les `read()` et `write()` bloquent uniquement le thread qui les appelle, laissant les autres threads travailler quand le kernel a terminé ses devoirs).
- C'est plus flexible et extensible (même si potentiellement tout aussi spaghetti que `poll()`/`select()`),
- On peut mieux contrôler les aspects temporels (temps de réponse, etc.).
- Les sémaphores sont des outils très puissants qui peuvent simplifier beaucoup de codes à base de mutex.
- Le découpage en tâches simples permet de passer facilement en multi-CPU, NUMA, MPI, cluster... L'avenir est parallèle !

## Pour la suite :

Le code multithread va servir de base pour les prochains articles sur la compression de données. Il est facilement extensible et peut même être adapté pour faire un lecteur ou un enregistreur de sons (ou les deux).

Pour expérimenter avec de nouvelles « charges », il suffit simplement de remplacer le code dans la boucle principale du `main()`. Des programmes similaires avec plus de threads pourront simuler un environnement multipiste.

Mais que l'on utilise les threads ou non, nos algorithmes de traitement vont traiter des blocs de taille variable au lieu de prendre une paire d'échantillons à la fois.

Ces algorithmes peuvent être inclus dans le code examiné ici ou dans un code plus simple sans threads et la question des détails techniques de bas niveau ne se pose plus : nous pouvons maintenant nous concentrer sur les traitements eux-mêmes.



**LIEN**

Miroir des sources : **http://ygdes.com**

**Yann Guidon,**

**whygee@f-cpu.org**
**http://ygdes.com**

**ANNEXE**

Pour obtenir les graphes de cet article, il a suffi de quelques utilitaires simples comme `vmstat`, `grep`, `cut`, `awk`, `gnuplot` et de quoi éditer les images.

### Collecte des statistiques

Dans un terminal séparé (alt-F1), il suffit de lancer `vmstat` et de rediriger la sortie vers un fichier. `tee` permet de conserver l'affichage en temps réel pour voir comment les choses se présentent sans attendre la fin de la mesure.

```
$ vmstat 1 |tee fichier.v
```

Normalement, `vmstat` ne devrait pas influencer les mesures, ou imperceptiblement, puisque ce programme a été conçu pour profiler le système. Puis, dans un autre terminal (alt-F2), lancer le programme à mesurer. Il faut aussi s'être assuré préalablement que le noyau n'a pas conservé des données partielles des expériences similaires précédentes. Un moyen est de remplir son cache avec des données inutiles, en lisant un fichier au moins deux fois plus gros que la mémoire de l'ordinateur. `dd if=/dev/zero of=/dev/null bs=1024K count=(2xtaille_RAM)` devrait faire l'affaire. De même, le fichier destination (quand il y en a) ne doit pas déjà exister, afin de forcer la réallocation des inodes (autant pousser le réalisme dans les moindres détails).

### Traitement des données

Deux choses à savoir sur `vmstat` : la première, indiquée dans la page man, est que la première ligne est la moyenne des paramètres **depuis le démarrage de l'ordinateur**, il faut donc l'enlever. La deuxième, qui n'est pas indiquée, est que le programme peut « rater » ou « sauter » des lignes si la charge CPU est beaucoup trop élevée, il faut alors considérer le résultat comme non fiable. Ensuite il faut un peu nettoyer le fichier car il contient des lignes de rappel du nom des colonnes, utiles pour nous les humains mais indésirables pour nos autres outils.

```
$ grep -v sy fichier.v > fichier.vms
```

Cela enlève les lignes contenant la chaîne `sy`, commune aux deux lignes ajoutées automatiquement. Il faut ensuite isoler les colonnes de nombres : c'est `cut` qui va s'en charger :

```
$ cut -b 51-56 fichier.vms (retourne la colonne "bo")
```

Il faut faire attention car les colonnes peuvent se décaler lorsque des nombres dépassent la capacité que `vmstat` leur alloue (comme lors de swap in/out par exemple). Un petit contrôle visuel préliminaire s'impose donc. Pour éviter ce désagrément, il aurait fallu un script en `sed` pour enlever les espaces superflus, par exemple : `sed 's/[ ][ ]/ /'`. J'aurais ainsi pu utiliser un autre mode de `cut` (champs séparés par des tabulations), mais j'ai choisi la route directe, les commandes commençant à devenir nombreuses et illisibles :-) En plus, cela aurait détruit l'alignement des nombres. Ensuite il faut continuer la préparation car `gnuplot` a un affichage inadapté à mes mesures en mode « ligne ». L'astuce pour faire les marches d'escalier est de dupliquer chaque point mais en décalant d'un cran l'indice. On obtient donc une ligne verticale et une ligne horizontale. C'est `awk` qui va s'en charger, puisque je n'ai pas réussi à le faire avec `sed`...

```
$ cut -b 51-56 fichier.vms | \
  awk '{print NR-1 $0; print NR $0}' > fichier.bo.dat
```

Pour décoder cet uniligne, il faut savoir que `NR` est la variable du numéro de ligne courante et `$0` est son contenu. Pour chaque ligne (les accolades étant un groupe de commande sans condition), `print` affiche donc deux fois la ligne avec les deux indices différents désirés. Comme il faut découper plusieurs colonnes et créer plusieurs fichiers, toutes les commandes sont dans un *script shell*. Une variable donne le nom de base du fichier à traiter pour éviter de le modifier à chaque fois. Bien sûr, il aurait été possible d'effectuer ces opérations avec un seul langage (j'entends d'ici des Mongueurs uniligner), mais je suis pour la diversité culturelle et la fainéantise unixienne :-)

### Mise en forme

Pour l'instant, je n'ai pas trouvé mieux que `gnuplot` (qui, rappelons-le, n'est pas un projet `GNU`). Il faut un peu tâtonner au début, mais on arrive rapidement à un résultat montrable. Voici par exemple le code d'une des illustrations :

```
# LECTURE+ECRITURE NORMALE
set key 83,21700
set xrange [-2:95]
set yrange [-500:20000]
set ylabel "KB" 3,25
set label "Fig. 5a" at 10,21000 c
set label "18000" at -6,18000 c
set label "trafic bidirectionnel" at 45,20500 c
set title "cat tetouatator.raw > t2.raw (1 minute 17s)"
plot "rw.bi.dat" t "lecture" with lines,\
     "rw.bo.dat" using ($1-0.3):($2) t "écriture" with lines
unset label
pause -1
```

Pour améliorer la lisibilité des lignes (qui ont tendance à se recouvrir les unes les autres), j'ai décalé certains tracés. C'est la fonction des expressions du type `($1-0.3):($2)` : on ajoute 0,3 au X et on laisse le Y tel quel.

### Finition

Capture d'écran, découpage, correction des couleurs. Tout le reste a déjà été fait par les programmes précédents. A vous de jouer ! Une bonne illustration vaut mieux que des mots et vos présentations n'en seront que plus attrayantes, lisibles et crédibles !

# → Introduction au Coldfire 5282

Jean-Michel Friedt, Simon Guinot,

**EN DEUX MOTS** **Cela commence par « Est-ce que ce code simple est aussi rapide que possible ? » et nous embarque dans « Quelle est la méthode la plus performante ? ».**

## 1 Introduction

Nous utilisons une carte commercialisée par la société allemande SSV [1] : la DIL/Net 5280. Cette carte fournit un processeur cadencé à 66 MHz, avec 16 MB de RAM et 8 MB de mémoire flash, et de nombreux périphériques que nous allons présenter plus tard dans cet article, pour un prix unitaire de 150 euros (250 euros avec une carte d'évaluation permettant une mise en œuvre en quelques minutes, 99 euros pour 10 pièces). Nous présenterons ici la mise en œuvre de cette carte, les divers périphériques que nous y avons ajoutés pour finalement conclure sur une application de capture d'images couleur retransmises par internet.

## 2 Présentation générale du matériel

La carte DIL/Net 5280 (figure 1) fournie sur circuit d'évaluation se caractérise avant tout par sa simplicité de mise en œuvre : noyau préflashé avec de nombreuses commandes unix classiques, un client NFS se connectant directement sur un serveur tel que installé sur un PC sous Debian (i. e. NFS v.3), un *bootloader* (dBug [2]) très puissant permettant de récupérer un nouveau noyau rapidement par tftp et de modifier l'adresse IP de la carte sans avoir à reflasher un noyau complet. SSV a installé un *shell* souple d'utilisation avec rappel des dernières commandes. En parallèle avec cette excellente qualité des logiciels sélectionnés, le circuit est simple à mettre en œuvre du point de vue matériel par l'utilisation d'un support DIL64 dont le brochage est très clairement documenté. Ainsi, contrairement à la carte uCdimm 5272 présentée auparavant [3], aucune compétence spéciale de soudure n'est requise pour faire fonctionner le DIL/Net 5280. Les principaux signaux utiles sont mis à disposition de l'utilisateur : bus de données du processeur, 4 lignes du bus d'adresse et un décodeur interne de plages d'adresses (*Chip Select*), divers bus de communication tels que I²C, SPI, RS232, Ethernet et CAN. Enfin, la mise sous tension de la carte 5282 ne lance pas nécessairement le bootloader mais permet de passer directement à l'exécution du noyau Linux placé en mémoire flash en fonction de la position d'un *jumper*, rendant le système facilement autonome dès son allumage.

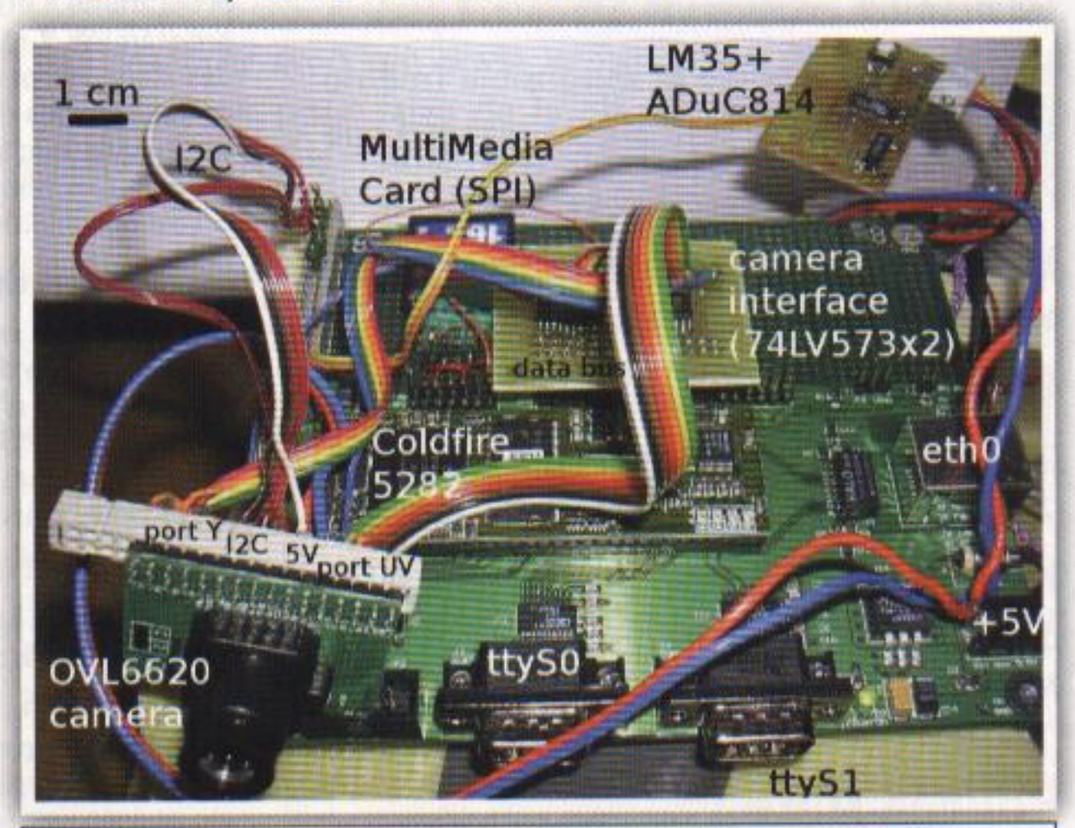



*Fig. 1 : Photographie de la carte SSV DIL/Net 5280 sur son support de développement équipé des connecteurs et de l'électronique d'interfaçage d'une mémoire de stockage (MultiMediaCard, section 5.1) de masse et d'une caméra numérique CMOS couleur (section 5.3. L'ADuC814 est un microcontrôleur 8-bits compatible 8051 utilisé ici pour tester la lecture sur le port série (cf. section 5.2).*

Malgré la ressemblance en termes de nomenclature avec le processeur Coldfire 5272 présenté antérieurement [3], une différence majeure du Coldfire 5282 est l'existence d'un mode protégé pour accéder aux registres matériels. Ainsi, contrairement au 5272 qui se programmait comme un microcontrôleur en accédant à l'ensemble de la mémoire depuis le mode utilisateur, la programmation du 5282 ressemble beaucoup plus à ce dont nous avons l'habitude sur un PC, avec l'utilisation intensive de modules noyaux pour toutes les opérations nécessitant un accès aux périphériques matériels. La transition du 5272 au 5282 est rendue aisée par la mise à disposition par SSV à l'achat de leur carte d'évaluation d'un module noyau nommé « ssvhwa.o » (fourni avec ses sources) permettant un accès simple (en 8, 16 ou 32 bits, entrée ou sortie) à toutes les adresses mémoire du 5282, y compris les registres d'accès aux ports matériels. En l'absence de ce module chargé de traduire des requêtes vers les registres de commande du matériel depuis le mode utilisateur, il nous faudra systématiquement écrire un module noyau pour accéder aux périphériques. ssvhwa se résume en fait en un bon outil de débuggage mais dont le temps d'accès aux registres matériels est tellement long qu'il nous faudra en pratique toujours recourir à l'écriture de module noyau lorsque des performances doivent être obtenues tel que nous le verrons dans les exemples illustrant cet article. En comparaison du 5272, la carte à base de 5282 garde les périphériques les plus communs : deux ports Ethernet, deux ports série, port SPI, PWM. La carte SSV n'embarque pas les convertisseurs

3,3 V→±12 V et transformateurs Ethernet mais reporte ces accessoires sur le circuit développé pour l'application spécifique (par exemple la carte d'évaluation SSV) : ainsi, cette carte est mieux appropriée pour communiquer avec des nombreux périphériques tels que microcontrôleurs et récepteurs GPS (qui requièrent généralement des connexions unipolaires sur leurs ports série) et évitent d'embarquer des transformateurs encombrants si la communication Ethernet n'est pas nécessaire. Dans le cas de la carte d'évaluation SSV (DNP/EVA6), la conversion 3,3 V→±12 V est obtenue au moyen de ADM3311 et le transformateur Ethernet est un HALO TG110-S050N2. Des périphériques peu utiles (esclave USB) sont avantageusement remplacés par des interfaces pratiques : I²C et convertisseurs analogique/numérique par exemple. Nous allons ici décrire l'accès à ces divers périphériques.

# 3 Outils de développements logiciels

Les outils de compilation sont les mêmes que pour 5272 (m68k-elf) et la même arborescence de binaires peut être utilisée avec le 5282. La grande nouveauté est l'utilisation intensive des modules noyaux pour lesquels le `Makefile` diffère un peu de celui utilisé pour la compilation de programmes utilisateurs : un exemple est fourni ici pour présenter les diverses options spécifiques à la compilation d'un module.

```
PATH := $(PATH):/usr/bin:/usr/local/bin
CC = m68k-elf-gcc
OBJS = i2c_mod.o
MODUL = -nostdinc -D__KERNEL__ -Wall -Wstrict-prototypes -Wno-
trigraphs -fno-strict-aliasing -fno-common \
-I/usr/local/lib/gcc-lib/m68k-elf/2.95.3/./include -pipe -DNO_MM
-DNO_FPU -m5307 -Wa,-S -Wa,-m5307 -D__ELF__ \
-DMAGIC_ROM_PTR -DUTS_SYSNAME="uClinux" -D__linux__ -DMODULE -I
$(UCPATH)/linux-2.4.x/lib/zlib_deflate \
-nostdinc -iwithprefix include -DKBUILD_BASENAME=deflate_syms  -
DEXPORT_SYMTAB
UCPATH = /home/jmfriedt/uClinux-dist

all: $(OBJS)

i2c_mod.o: i2c_mod.c
 $(CC) $(MODUL) -O3 -c i2c_mod.c -I$(UCPATH)/linux-2.4.x/include

# NE PAS METTRE -msep-data dans un module
```

# 4 Spécificités de la programmation sur système embarqué

Un certain nombre de différences existent entre le développement sur PC sous GNU/Linux et sur les systèmes embarqués fonctionnant sous uClinux. Nous allons présenter ici quelques aspects que nous avons abordés au cours de nos expériences, notamment concernant la gestion de la mémoire et des interruptions dans cet environnement particulier.

## 4.1 Rappels sur les modules

Les modules deviennent une part fondamentale de la programmation des périphériques matériels du Coldfire 5282 qui ne sont accessibles qu'en mode superviseur, i. e. mode noyau pour Linux. La structure générale des modules sous uClinux est strictement identique à celle familière sous Linux. La seule originalité de la programmation sur Coldfire 5282 dans cet environnement de module noyau est la présence de lignes telles que :

```
*((volatile u8 *)addr)=data; ou
data = *((volatile u16 *)addr);
```

qui correspondent respectivement à des écritures et lectures à des adresses prédéfinies telle que décrites dans la documentation du Coldfire 5282 [4] pour un accès aux périphériques matériels.

## 4.2 Les interruptions

Le Coldfire 5282 dispose de deux contrôleurs d'interruption. Chacun de ces contrôleurs comprend 63 vecteurs dont 56, pleinement configurables et programmables. La table des vecteurs d'interruptions comporte 256 entrées. Les 64 premiers vecteurs sont réservés au fonctionnement interne du processeur (gestion du reset, gestion des erreurs...). Le premier contrôleur permet de configurer les vecteurs 65 à 127 et le second, les vecteurs 128 à 190. Ces deux contrôleurs sont appelés respectivement INTC0 et INTC1. Ils peuvent être configurés par l'intermédiaire de registres mappés en mémoire. Pour INTC0 par exemple, l'adresse de base est IPSBAR + 0xc00 et pour INTC1, elle est IPSBAR + 0xd00. Nous allons présenter les registres dont les fonctionnalités sont les plus intéressantes, puis nous illustrerons l'activation d'une interruption au travers d'un exemple.

### 4.2.1 Concurrence entre les interruptions

La concurrence entre les interruptions est un point critique des applications temps réel. Sur les processeurs de type Motorola Coldfire, il est possible de classer les vecteurs d'interruption en fonction de leur importance. Dans le cas du Coldfire 5282, le registre 8 bits ICR (*Interrupt Control Register*) va permettre d'assigner un niveau et une priorité à une interruption. Chaque vecteur d'interruption dispose de son propre registre ICR. Dans une situation de concurrence entre plusieurs interruptions, celle dont le niveau est le plus important sera exécutée en premier. Si plusieurs d'entre elles possèdent le même niveau, elles sont départagées en examinant leurs priorités. Les priorités et niveaux sont désignés par des valeurs comprises entre 0 et 7 (3 bits). Une interruption de niveau élevé sera également autorisée à préempter le gestionnaire associé à une interruption de niveau moindre. Pour éviter cela, les applications temps réel devront adjoindre une forte priorité à leurs vecteurs d'interruption.

À noter que le système de priorité n'existe pas sur le Coldfire 5272. Le registre PIWR (*Programmable Interrupt Wakeup Register*) est utilisé pour départager les interruptions de même niveau. Le vecteur représenté par le bit de poids le plus fort au sein de ce registre est acquitté le premier. Il est évident que ce type de configuration est bien moins souple à gérer pour le programmeur...

### 4.2.2 Masquer les interruptions

Chaque contrôleur d'interruption dispose de 2 registres de 32 bits IMRH et IMRL (*Interrupt Mask Register Hight or Low*). Chacun des bits de ces registres est associé à un vecteur d'interruption du contrôleur. Positionner à 1 le bit correspondant à un vecteur d'interruption permet de masquer ce dernier. Il est également possible d'utiliser le registre global SR (*Status Register*) qui met 3 bits à disposition des interruptions. Ces 3 bits permettent de définir un niveau. Toutes les interruptions de niveau inférieur à la valeur stockée dans ce registre sont masquées. À noter que les fonctions assembleur `cli()` et `sti()`, permettant respectivement de masquer et démasquer toutes les interruptions, utilisent ce registre. L'idée est d'y inscrire la valeur 7, correspondant au niveau le plus important. Cela a pour effet de masquer toutes les interruptions, exceptées celles de niveau 7. Il n'existe aucune méthode pour masquer ces dernières. Le niveau 7 regroupe les interruptions internes considérées comme critiques pour le bon fonctionnement du processeur (gestion des différentes erreurs : opérations illégales, accès mémoire interdits... entre autres). Implémentation de la fonction `cli()` telle que trouvée dans le fichier `/uClinux-dist/linux-2.4.x/include/asm/system.h` :

```
#define __cli() __asm__ __volatile__ ( \
        "move %/sr,%%d0\n\t" \
        "ori.l  #0x0700,%%d0\n\t" \
        "move %%d0,%/sr\n" \
        : /* no inputs */ \
        : \
        : "cc", "%d0", "memory")
#else
```

### 4.2.3 Les interruptions logicielles

Les registres 32 bits INTFRCH et INTFRCL (*Interrupt Force Register Hight or Low*) sont également disponibles sur chacun des contrôleurs. Chaque bit de ces registres est associé à un unique vecteur d'interruption. Positionné ce bit à 1 permet de générer de manière logicielle une interruption du vecteur correspondant. Ces registres sont particulièrement utiles lors du développement et du debuggage de pilotes.

Exemple de fonction permettant sur le Coldfire 5282 d'activer un vecteur d'interruption et de lui associer un gestionnaire :

```
struct irq_info
{
        unsigned int vector;
        unsigned char name[30];
        unsigned char level; /* level and priority requested */
        void (*handler)(int, void *, struct pt_regs *);
        void *dev_id;
};
static int enable_interrupt_vector (struct irq_info *irq_info)
{
        unsigned int retval = 0;
        unsigned int int_num;
        volatile unsigned char *icr;
        volatile unsigned long *imrh;
        volatile unsigned long *imrl;
        [...]
```

`int_num` et les pointeurs `icrp`, `imrh` et `imrl` sont initialisés en fonction du numéro de vecteur `irq_info→vector` fournit en paramètre. Puis le gestionnaire désigné par `irq_info→handler` est associé à ce vecteur à l'aide de la fonction `request_irq()`. Cette phase d'enregistrement est identique à celle qui se pratique sur un système GNU/Linux classique.

```
    if ((retval = request_irq (irq_info->vector, irq_info->handler,
                               SA_INTERRUPT, irq_info->name,
                               irq_info->dev_id)))
    {
            dbg("unable to request irq %d", irq_info->vector);
            return retval;
    }
```

Le registre `ICR` correspondant au vecteur que l'on souhaite activer se voit affecter un niveau et une priorité. Cette information est également fournie en paramètre via `irq_info→level`.

```
icr[int_num] = (irq_info->level > IRQ_LEVEL_MAX)?
                IRQ_LEVEL_DFLT : irq_info->level;
```

Et enfin, on s'assure que le bit de masque correspondant a notre vecteur n'est pas positionner.

```
        *imrh &= ~(1 << (int_num - 32));
        *imrl &= ~1;
```

À partir d'ici, le vecteur d'interruption est actif et la fonction `irq_info→handler` lui est associée.

```
        [...]
        return (retval);
}
```

## 4.3 Gestion de la mémoire sur un système sans MMU (Memory Management Unit)

La MMU correspond à l'ensemble des mécanismes internes au processeur permettant de réaliser la conversion d'une adresse logique en une adresse physique. Ces mécanismes peuvent être la pagination, la segmentation ou encore la pagination de segments... Une conséquence directe de l'absence de MMU est l'impossibilité pour uClinux d'implémenter un VM (*Virtual Manager* ou gestionnaire de mémoire virtuelle). Le rôle du VM est de permettre aux processus de mapper des zones de mémoire physique non contiguës au sein de leur espace d'adressage linéaire. C'est sur le VM que repose le partage des librairies. Cette technique permet à plusieurs processus

d'utiliser une librairie commune en la mappant dans leur espace d'adressage virtuel. De cette façon, la librairie n'est chargée qu'une seule fois en mémoire physique. L'absence de VM rend impossible l'utilisation de librairies partagées. Toutes les applications à destination d'uClinux sont donc compilées statiquement et les librairies nécessaires à leur fonctionnement font parties intégrantes du binaire final. Lors du chargement en mémoire du programme, ces librairies seront copiées intégralement dans son espace d'adressage. Il en résulte un problème de redondance. En effet, les librairies les plus couramment utilisées, comme `uClibc` (version allégée et statique de la `libc`), seront présentes plusieurs fois en mémoire physique. Le VM permet également de contrôler les accès mémoire et les permissions. Sur un système GNU/Linux classique, un accès mémoire illégal pourra engendrer une erreur de segmentation ou de pagination : le fameux « *segmentation fault* » bien connu des programmeurs. L'absence de tels mécanismes sur les processeurs sans MMU prive uClinux de contrôle sur les accès mémoire. Un processus pourra donc accéder et écrire à des adresses encore non allouées ou même extérieures à son espace d'adressage. Prenons comme exemple un programme dont un tampon déborde légèrement et qui, la majeure partie du temps, s'exécute normalement. À l'issue d'un changement de contexte, on peut cette fois imaginer le tampon écraser des données importantes et causer une erreur. Ce genre de bug est particulièrement délicat à repérer et est le plus souvent entouré d'un épais mystère. C'est pourquoi les programmeurs devront être particulièrement vigilants vis à vis des accès mémoire sous uClinux. Dans le cas du Coldfire 5272, l'absence de contrôle des accès mémoire présentait aussi des avantages. Les programmes utilisateurs étaient capables d'accéder à des registres normalement réservés au noyau. Il était ainsi possible d'activer un certain nombre de fonctionnalités matérielles depuis l'espace utilisateur. L'idée n'était pas de développer des pilotes en espace utilisateur mais plutôt de tester des solutions avant de se lancer dans le développement assez fastidieux d'un module noyau plus rapide et plus performant. Le 5282 disposant d'un mode superviseur, justement destiné à protéger les registres critiques, ne nous offre plus ce petit avantage. Les registres protégés seront donc uniquement adressables en mode noyau.

### 4.3.1 Absence de fork

L'API uClinux est relativement compatible avec celle de GNU/Linux... à quelques exceptions près. L'absence de l'appel système `fork()` en est une bien connue. Encore une fois, il s'agit d'une conséquence de l'absence de VM. Les implémentations modernes du `fork()` utilisent la stratégie du *copy on write*. À l'issue de l'appel `fork()` un processus fils est créé et son espace d'adressage virtuel pointe sur le même espace mémoire physique que celui de son père. Les premiers accès en écriture vont engendrer la création de pages séparées pour le père et le fils. L'idée est de retarder le plus longtemps possible l'opération de duplication de l'espace d'adressage du processus père. Cette stratégie rend impossible le port de l'appel système `fork()` sur un système dépourvu de MMU. Comme alternative uClinux offre le vieil appel système BSD `vfork()`. Historiquement, `vfork()` a été écrit comme une alternative aux premières implémentations de `fork()`. Ces dernières n'utilisaient justement pas le copy on write et allouaient directement à la création d'un processus fils un espace distinct et le contenu de celui du père y était dupliqué. Cette opération, relativement coûteuse en temps, s'avérait souvent inutile. Particulièrement lorsque après sa création, le processus appelait une fonction du type `execve()`, provoquant l'exécution d'un nouveau programme et toutes les allocations mémoire qui en découlent. `vfork()` permet au père et au fils de partager l'intégralité de leur espace d'adressage, y compris la pile. Pour éviter tout conflit, le père voit son exécution suspendue jusqu'à ce que le fils appelle une fonction de type `execve()` ou `_exit()`. Cet appel système est assez rapide mais ne permet pas, hélas, d'émuler plusieurs processus concurrents au sein d'une même application. Les restrictions sont donc relativement importantes et certaines applications ne sont pas trivialement portables sous uClinux.

### 4.3.2 Pile de taille fixe

Les applications fonctionnant sous uClinux disposent d'une pile de taille fixe et non extensible qui est allouée au chargement en mémoire du programme. Sur le Coldfire 5282, la taille par défaut de la pile est de 16 MB. Cette taille peut être fixée pendant la compilation en ajoutant la ligne `FLTFLAGS = -s 4096` au Makefile de l'application. Une autre méthode pour modifier la taille de la pile d'un binaire existant est d'utiliser l'outil `m68k-elf-flthdr` là encore avec l'option `-s`. Si on cumule le fait que la pile n'est pas dynamiquement extensible et que les accès mémoire ne sont pas contrôlés, il devient évident que le programmeur devra être très attentif quant à la gestion de la pile. Un programme dont la pile est en train de déborder ne générera pas d'erreur d'accès mémoire. Au mieux, elle débordera sur des zones mémoire encore non allouées et le programme s'exécutera normalement. Dans le pire des cas, l'espace d'adressage d'un processus voisin sera altéré et ce dernier pourra en voir son fonctionnement modifié. En règle générale, les fonctions récursives doivent être surveillées et les allocations dynamiques être préférées à celles sur la pile.

### 4.3.3 L'allocation dynamique de mémoire

Un problème associé à l'absence de gestionnaire de mémoire sur les processeurs exécutant le noyau uClinux est la limitation quant à l'allocation d'une grande quantité de

mémoire telle que requise lors de l'acquisition d'un grand nombre de données. Sur un système d'exploitation disposant d'un VM (gestionnaire de mémoire virtuelle), les processus allouent dynamiquement de la mémoire dans le tas (*heap*). Ces allocations dynamiques se font à l'aide de fonctions ou d'appels système mis à disposition par le système d'exploitation. Sous GNU/Linux, l'appel système `brk()` ou la fonction de plus haut niveau `malloc()` (elle-même utilisant `brk()` en interne) peuvent être utilisés. Le tas correspond à un espace contigu d'adresses linéaires contenu dans l'espace d'adressage du processus. Le VM permet de mapper des adresses physiques non contiguës au sein de cet espace d'adresses linéaires. La taille du tas peut être dynamiquement augmentée si le besoin se fait sentir, en élargissant l'espace d'adresses linéaires lui étant attribué. La plupart des systèmes d'exploitation sans VM implémentent un tas de taille fixe et non extensible, résidant dans l'espace d'adressage du processus. Cette méthode a l'inconvénient de gaspiller inutilement de la mémoire si cet espace n'est pas utilisé par le processus et aussi de rendre impossible les allocations mémoire trop volumineuses (*i. e.* dépassant la taille totale prévue pour le tas). uClinux contourne cette difficulté en utilisant une stratégie originale : les processus ne disposent pas de tas individuel [12]. À la place, les allocations dynamiques sont satisfaites en piochant dans un *pool* de mémoire correspondant à l'ensemble de la mémoire libre du système.

Grâce à cette stratégie, un processus peut, s'il en a le besoin, s'attribuer l'ensemble de la mémoire disponible sur le système. Un problème peut cependant survenir dans le cas de fortes allocations mémoire. En effet, la fragmentation de la mémoire physique va souvent rendre impossible les allocations importantes de mémoire contiguë. Dans l'exemple présenté figure 2, schématisant 1 MB de mémoire physique, on peut remarquer que l'on dispose de 500 KB de mémoire disponible. Cependant, une allocation de 100 KB échouera. En raison de la fragmentation, une telle plage d'adresses mémoire contiguës n'est pas disponible.

Le problème de la fragmentation de la mémoire n'a pas vraiment de solution satisfaisante. Les allocations importantes devront être réalisées rapidement après le démarrage du système d'exploitation, au moment où la mémoire est la moins fragmentée. Pour une application ne nécessitant pas impérativement que la mémoire allouée soit contiguë, on peut envisager d'utiliser une liste chaînée. Chaque élément de la liste pointe vers une zone mémoire allouée par le processus et aussi vers l'élément suivant. Ce dernier désignera une autre zone mémoire, etc.

```
struct buffer
{char *data;
 unsigned int size;
 struct buffer *next;
};
struct buffer * alloc_mem (unsigned int size, unsigned int *get_
size);
```

Si l'on reprend l'exemple précédemment illustré figure 2, où l'allocation de 100 KB était impossible... elle pourra être réalisée à l'aide de cette liste, avec par exemple deux éléments, le premier pointant sur une zone de 80 KB et le suivant sur une zone 20 KB. Cette astuce ne pallie pas l'absence de mémoire contiguë, mais elle permet de stocker des zones de mémoire non adjacentes dans une structure de données simple à manipuler.

## 5 Applications pratiques

Nous allons présenter ici quelques applications développées pour nous familiariser avec ce circuit, de complexité et de fonctionnalités croissantes :

- Écriture en entrée sortie sur un port numérique par le module `ssvhwa` puis par notre propre module ;
- Le stockage de données sur mémoire non volatile de type MultiMediaCard (MMC, http://www.sandisk.com/oem/mmc.asp) ;
- La conversion analogique/numérique avec un échantillonnage lent ;
- L'acquisition d'images couleur via le bus de données et configuration de la caméra par bus $I^2C$ Figure 4.

### 5.1 Le port SPI et la MultiMediaCard (MMC)

De nombreuses méthodes de stockage de masse non volatiles sont aujourd'hui disponibles à très faible coût du fait de leur utilisation dans l'électronique grand publique : cartes SmartMedia, Memory Stick, CompactFlash Multimedia (MultiMediaCard, MMC) pour ne citer que les plus courantes. Nous avons sélectionné le dernier type pour son faible nombre de broches (connecteur 7 broches séparées de 2.54 mm – un bout de connecteur ISA peut être utilisé pour s'interfacer), sa simplicité de programmation tel que

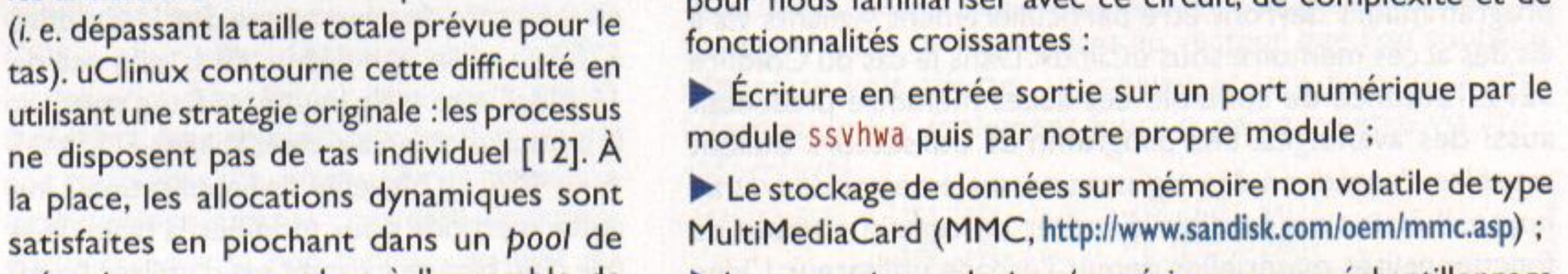



*Fig. 2 : Exemple de fragmentation d'une mémoire de 1 MB : alors que 500 KB sont disponibles, toute tentative d'allocation de mémoire de 100 KB échoue.*

nous allons le voir plus loin, son faible coût pour une zone de stockage importante (16 MB à 128 MB pour moins de 15 euros) et sa disponibilité en volume unitaire. Nous allons développer en détail le mode de programmation de ces cartes car au-delà de uClinux tournant sur Coldfire, nous avons implémenté ce protocole sur de nombreuses autres architectures (microcontrôleurs 8 bits compatible 8051 Analog Devices ADuC814, Motoroloa 68HC908JB8, accès depuis le port parallèle d'un PC compatible IBM), ouvrant des possibilités de stockage de télémesure inimaginables sinon. Nous allons ici nous focaliser sur l'aspect QSPI du Coldfire 5282, une description des autres implémentations étant disponible sur la page web des auteurs ou dans la littérature [7, 8, 9]. La MultiMediaCard s'initialise par défaut dans un mode de communication synchrone spécifique à cette carte. Nous avons décidé de profiter de la disponibilité d'un port SPI sur le processeur Coldfire (comme sur la majorité des microcontrôleurs Motorola/Freescale) pour utiliser ce mode de communication que comprend la MMC après une phase d'initialisation appropriée. Nous allons commencer par décrire l'accès au bus SPI du Coldfire en général, pour ensuite nous focaliser sur le cas particulier de la MMC [10].

### 5.1.1 Le port SPI

Le bus SPI supporte un protocole bidirectionnel synchrone ne nécessitant que 3 fils : une horloge, une communication du maître vers l'esclave (MOSI : *Master Out Slave In*, du Coldfire vers la MMC dans notre cas) et de l'esclave vers le maître (MISO : *Master In Slave Out*). Le Coldfire 5282 est équipé d'une QSPI, *i. e.* une version évoluée du port SPI incluant une pile de 64 mots permettant au logiciel de gestion de la communication une plus grande latence entre deux accès au bus. Nous allons dans notre cas généralement nous limiter à n'utiliser que le premier élément de cette pile, de façon très semblable au mode d'accès sur microcontrôleur. Une première phase d'initialisation active le port SPI (registre PQSPAR=0x0f), définit le protocole de communication à 8 bits/donnée à une fréquence de l'ordre de 100 kHz (registres QMR=0xA1A2, QDLYR=0x0202 et QIR=0xD00F pour désactiver toute interruption associée au port SPI).

### 5.1.2 Cas particulier de la MMC

Le passage du mode de communication parallèle MMC au mode série SPI se fait en envoyant à l'allumage de la carte une trame de 80 créneaux sur le signal d'horloge tout en maintenant le niveau du signal CS# de la MMC haut (i. e. désactivé) [11]. Une pile de commande est donc initialisée à cette fin en remplissant les 10 premiers éléments du registre QDR et en définissant le registre QAR=0x20 pour indiquer que 2×10 commandes sont à exécuter. La suite des opérations est décrite de façon claire par le code source : envoi de la commande `CMD0` (Reset : valeur 0x40 transmise du Coldfire vers la MMC avec CS# actif au niveau bas), puis attente de la transmission d'une valeur 1 par la MMC pour acquitter le passage au protocole SPI. L'initialisation de la MMC s'obtient ensuite par transmission de la commande `CMD1` (valeur 0x41 transmise du Coldfire vers la MMC avec CS# actif au niveau bas) avec une somme de contrôle (`checksum`) prédéfinie à 0x95 et acquittement de la MMC par une réponse de 0. Notez que toutes les commandes `CMDxx` décrites dans les documentations de MMC correspondent à l'envoie de la valeur `xx` en décimal à la carte, masquée par un `OU` logique avec la valeur `0x40`. Par exemple la transmission d'un ordre d'écriture de bloc (commande `Write_Block CMD24`) s'obtient par émission de l'octet `0x58` (`OU` logique entre `0x40` et `0x18` qui est l'expression de `0d24` en hexadécimal).

Une fois la MMC passée en mode SPI, le contrôle de `checksum` est désactivé et ce dernier sera toujours maintenu à `0xFF`. L'écriture en mode SPI sur la carte MMC se fait nécessairement par blocs de taille déterminée : cette taille est définie par défaut à 512 octets et nous la laisserons à cette valeur par la suite (nous pourrions modifier cette taille de bloc par la commande `Set_BlockLen`, `CMD16`).

Par conséquent toute adresse de bloc fournie comme point de départ d'une nouvelle lecture ou écriture est nécessairement multiple de 512, soit une adresse de la forme `0xab 0xcd 0x20*i 0x00 (i=0..7)`. La commande MMC d'écriture dans un bloc est `0x18` (commande `CMD24` en décimal dans la nomenclature MMC) qui prend pour argument l'adresse d'écriture et se conclut après remplissage du bloc (transfert de 512 octets) par un acquittement confirmant l'écriture des données sur la carte mémoire.

Un protocole similaire est appliqué pour la lecture de bloc– opération `0x11` (`CMD17` en décimal dans la nomenclature MMC : `Read_Single_Block`) avec là encore pour argument l'adresse du bloc de 512 octets à lire de la forme de celle vue auparavant. Ce mode de stockage a été utilisé avec succès lors de l'acquisition de données issues des convertisseurs analogique/numérique ou pour la sauvegarde de données synthétisées par notre programme. Le passage par la carte MMC selon un protocole que nous implémentons par nos propres soins a notamment l'avantage, par rapport aux appels système standards de type `write()` qui bloquent les interruptions, de n'interférer avec aucune interruption et par conséquent de permettre à une application de temps réel dur de s'exécuter en ne sauvant les données que lorsque le temps le permet.

## 5.2 Les convertisseurs analogique/numérique

Nous allons dans un premier temps illustrer une méthode relativement lente d'échantillonnage des convertisseurs analogique/numérique – de l'ordre de quelques échantillons par seconde au mieux pour des applications telles que l'observation de l'environnement par

exemple (température telle que présentée figure 3, pression) – l'acquisition rapide à des fréquences audio faisant l'objet d'un autre article du fait des diverses difficultés rencontrées.

Comme dans le cas du bus SPI, les convertisseurs du Coldfire 5282 sont accessibles via une pile (QADC). Là encore, nous chargerons dans un premier temps un certain nombre de commandes de conversion (mode de conversion et numéro de la voie à activer) pour ensuite lancer l'ensemble de ces commandes et enfin lire dans une seconde pile les résultats de conversion.

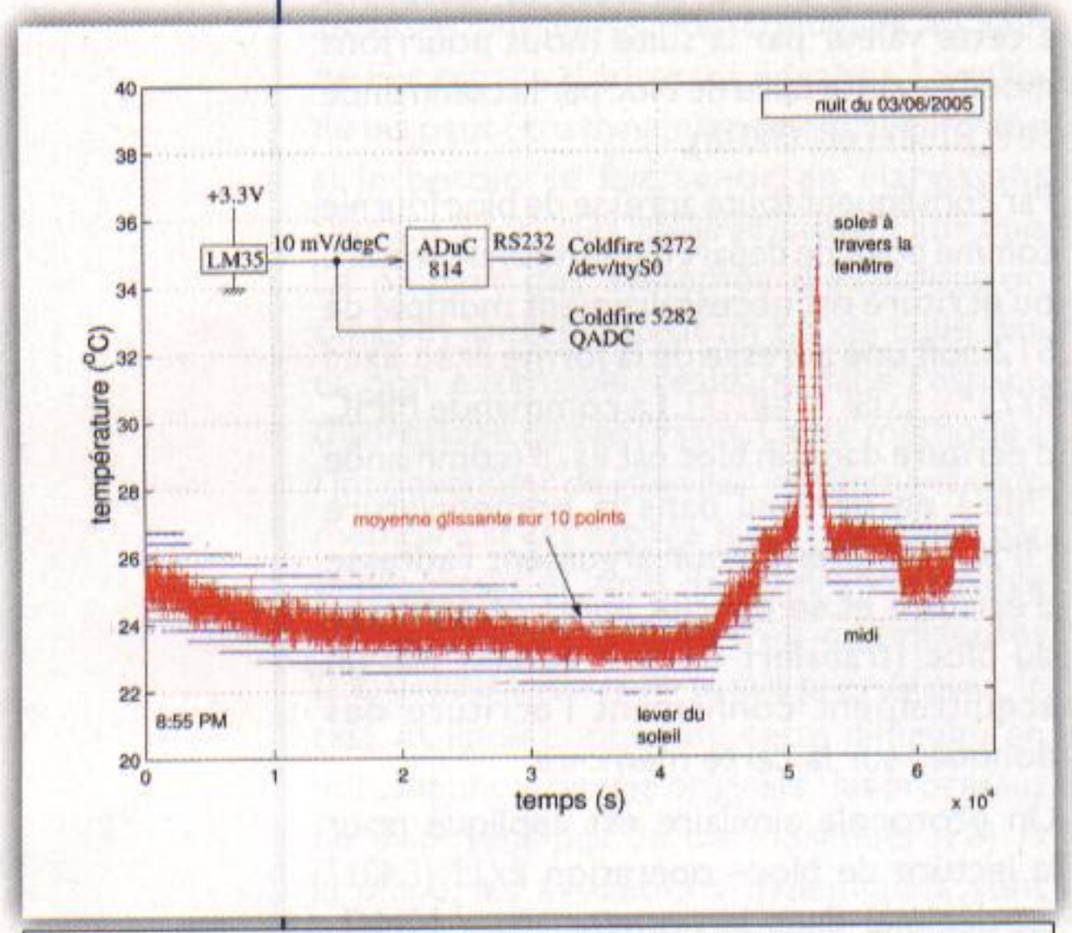



***Figure 3 : Exemple de mesure de température à 1 échantillon/s par le port QADC du 5282, issue sans traitement d'un capteur LM35 fournissant 10 mV/degrés C. Le rapport signal sur bruit est amélioré (et la résolution temporelle dégradée) par une moyenne glissante sur 10 points qui permet de faire disparaître la discrétisation par la conversion analogique/numérique.***

Un exemple de conversion à un échantillon par seconde est présenté sur la figure 3 tel que obtenu avec le programme ci-dessous :

```
#include <stdio.h>
#include <math.h>
#include "ssvhwa.h"

#define MCFBAR  0x40000000

#define QADCMCR  (MCFBAR + 0x00190000)  /* 16 bit */
#define DDRQA    (MCFBAR + 0x00190008)  /* 8 bit */
#define QACR0    (MCFBAR + 0x0019000A)  /* 16 bit */
#define QACR1    (MCFBAR + 0x0019000C)  /* 16 bit */
#define QASR0    (MCFBAR + 0x00190010)  /* 16 bit */
#define QASR1    (MCFBAR + 0x00190012)  /* 16 bit */
#define CCW      (MCFBAR + 0x00190200)  /* N bit */
#define RJURR    (MCFBAR + 0x00190280)  /* N bit */
#define LJSRR    (MCFBAR + 0x00190300)  /* N bit */
#define LJURR    (MCFBAR + 0x00190380)  /* N bit */

#define fichier
// sauvegarde dans un fichier => pas d'affichage

#define nbscan 1 // 64
int main (void)
{unsigned char data_b;
 unsigned short data_w,i;
#ifdef fichier
 FILE *f;
 f=fopen("adc.dat","w");
#endif
    // check user identity
    if (geteuid() != 0) {printf("No root access rights !\
n");exit(1);}
    if (ssvhwa_open() < 0) {perror("ssvhwa open");exit(-1);}
// set DIL/NetPC PIO Port A = output
 ssvhwa_write8(DDRQA, 0x00);      // Port A[0-3] output : 0x1B
 ssvhwa_write16(QADCMCR, 0x0000); // enable ADCs
 ssvhwa_write16(QACR0,0x7f);
// default val=19, QACR0 pour definir QCK
// 52=AN52=PQA0, 53=AN53=PQA1, 62=(VH-VL)/2
for (i=0;i<nbscan*2;i+=2) ssvhwa_write16(CCW+i,0x00C0|52);
// queuel read, 16 ck

while (1) {
 ssvhwa_write16(QASR0,0x0000);
 ssvhwa_write16(QACR1,0x2100);
// queue 1 single scan mode, software trig 2100
 do {data_w=(ssvhwa_read16(QASR0));
#ifndef fichier
  printf("QASR0=%x ",data_w);
#endif
 } while ((data_w&0x8000)==0);
#ifndef fichier
 printf("\n");
#endif
 for (i=0;i<nbscan*2;i+=2)
   {data_w=ssvhwa_read16(RJURR+i);printf("%x=%d /10=degC",data_
w,(int)((float)data_w*3.3/1.024));
#ifdef fichier
    fprintf(f,"%u\n",data_w);fflush(f);
#endif
  }
 printf("\n"); sleep(1); // 1 value/second
 }
 ssvhwa_close();return(0);
}
```

Ce programme se charge dans un premier temps d'activer les convertisseurs analogique/numérique (désactivés par défaut pour limiter la consommation électrique du Coldfire) en mettant à 0 le registre QADCMCR. Le principe des queues de conversion est qu'une table de 64 commandes est mise à disposition de l'utilisateur au moyen de la table de registres CCW. Nous y plaçons le mode de conversion (continu, unique, déclenché par un signal externe...) et la voie de multiplexage activée. Dans notre exemple, nous faisons une unique conversion immédiate sur la voie 52 (broche AN52 du Coldfire 5282). Après lecture du registre de statut (QASR0) qui nous annonce que la conversion a eu lieu, nous lisons le résultat de conversion dans un mot de 16 bits aligné à droite (i. e. dont seuls les 10 premiers bits sont significatifs) sous la forme du premier élément de la table de 64 mots nommée RJURR. Nous présenterons ultérieurement pour une application spécifique le cas de conversions multiples à fréquence d'échantillonnage élevée. Nous passons ici sous silence la disponibilité d'une méthode de division des tables de 64 mots en sous-ensembles (nommés « queue 1 » et « queue 2 ») dont l'intérêt nous échappe encore malgré des tentatives d'éclaircissement lors de communications avec les ingénieurs Freescale chargés de l'assistance des développeurs sur Coldfire. La conversion des valeurs lues en valeurs significatives physiquement se fait par tension= valeur× 3,3/1024 où 3,3 V est la tension de référence (tension d'alimentation du Coldfire) et 1024 la

résolution des convertisseurs (10 bits). Dans l'exemple qui nous intéresse ici, le capteur de température est un LM35 fournissant 10 mV/degré celsius.

## 5.3 La caméra couleur : les bus du processeur et bus I²C

Les capteurs optiques classiques de type CCD tendent aujourd'hui à être remplacés, pour de nombreuses applications où la sensibilité lumineuse n'est pas primordiale, par des capteurs CMOS. Ces derniers ont l'avantage d'intégrer dans un même composant le capteur optique analogique et l'électronique numérique de gestion de ce capteur pour finalement ne fournir à l'utilisateur qu'une interface totalement numérique, beaucoup plus simple à gérer que les nombreuses tensions et signaux d'horloge nécessaires pour une CCD [5]. Nous avons sélectionné, pour des raisons de coût et de disponibilité en petites quantités, le capteur CMOS OV6620 de Omnivision disponible monté sur circuit imprimé et équipé d'une optique chez Lextronic sous la dénomination CA88 [6]. Contrairement à la caméra Connectix Quickcam que nous avions décrite antérieurement [3], l'OV6620 n'est pas esclave des requêtes du processeur pour un nouveau pixel, mais impose sa fréquence d'horloge. Ainsi, le composant est cadencé par une référence de fréquence fixe, divisée en interne par une valeur programmable, pour fournir en sortie à intervalle de temps régulier des pixels. La vitesse par défaut, 50 images de 352×288 pixels par seconde, est trop élevée pour un traitement logiciel par un processeur généraliste comme le Coldfire. Notre première tâche consiste donc à comprendre la communication par protocole I²C pour programmer l'OV6620 et ainsi abaisser la vitesse de sortie des pixels.

### 5.3.1 Le protocole I²C

Le bus I²C supporte un protocole synchrone et ne nécessite que deux fils : une horloge (SC) et le transfert bidirectionnel de données (SD). Ce bus est utilisé – sous un nom (SCCB) qui n'enfreint pas le copyright de Philips sur le nom de ce bus – par les caméras Omnivision pour leur configuration. D'un point de vue matériel, le port I²C du Coldfire (3,3 V) se connecte au bus SCCB de l'OV6620 (5 V) en ajoutant des résistances de *pull-up* à 5 V (≈5,5 kΩ), ou directement au bus SCCB de l'OV3620 (elle aussi en 3,3 V) sans résistance de pull-up. Ayant décrit en détail l'accès au bus SPI auparavant, nous n'allons pas ici entrer dans les détails de l'accès au bus I²C qui est très similaire dans le principe. Un exemple de programme court se chargeant de l'initialisation de la caméra est fourni ici pour référence. Il utilise le module `ssvhwa` fourni par SSV pour un accès aux registres de contrôle matériel depuis l'espace utilisateur, mais pourra facilement être transféré dans un module noyau gérant l'initialisation de la caméra. Divers conditions permettent de sélectionner la vitesse de transfert des images, le choix du mode RGB ou YUV et le transfert de données en mode 8 bits (port Y uniquement) ou 16 bits (port Y et UV) :

```
#include <stdio.h>
#include "ssvhwa.h"
#define I2C 0xc0                        // CA88 base address
#define MCFBAR  0x40000000
#define I2ADR (MCFBAR + 0x300)
#define I2FDR (MCFBAR + 0x304)
#define I2CR (MCFBAR + 0x308)
#define I2SR (MCFBAR + 0x30C)
#define I2DR (MCFBAR + 0x310)
#define PASPAR (MCFBAR + 0x00100056)
// 16 bits, AS=mot de poids fort
void stop_i2c()
{ssvhwa_write8(I2CR, 0x90);}            // slave rcv mode
void start_i2c()
{unsigned char data_b;
 ssvhwa_write8(I2CR, 0xB0);             // master transmit mode
 do {data_b=ssvhwa_read8(I2SR);printf("I2SR=%x - ",data_b);}
    while ((data_b&&0x20)==0);   // IBB is not set = not yet busy
}
char snd_i2c(unsigned char b)
{unsigned char data_b;
 ssvhwa_write8(I2DR,b);                 // transmit data
 printf("%x: ",b);fflush(stdout);
 do {data_b=ssvhwa_read8(I2SR);printf("I2SR=%x\n",data_
b);fflush(stdout);} while ((data_b&0x80)==0);  // sent ?
 ssvhwa_write8(I2SR,0x00);          // clear "byte received" flag
 data_b=(ssvhwa_read8(I2SR)&0x01);
 return(data_b);            // returns 0 if success, 1 if failed
}
void snd_cmd(unsigned char reg,unsigned char val)
{start_i2c();
 while (snd_i2c(I2C)!=0) {stop_i2c();start_i2c();}
// dest @ (0=write)
 snd_i2c(reg);snd_i2c(val);
// destination register -- value written
 stop_i2c();
}
int main (int argc, char **argv)
{unsigned char data_b;
 unsigned long data_l;
 int duree=0x3F;
 // check user identity
    if (geteuid() != 0){printf("No root access rights !\n");
exit(1);}
    if (ssvhwa_open() < 0) {perror("ssvhwa open");exit(-1);}
// ENABLE I2C
   data_l=ssvhwa_read16(PASPAR); data_l|=0xffff;
   ssvhwa_write16(PASPAR,data_l);
   printf("PASPAR=%x\n",ssvhwa_read16(PASPAR));
    ssvhwa_write8(I2FDR,0x15);          // ck ~ 100 kHz
    ssvhwa_write8(I2ADR,0x42);          // slave @: not used
    ssvhwa_write8(I2CR,0x00);           // reset I2C
    ssvhwa_write8(I2CR,0xA0);
    data_b=ssvhwa_read8(I2DR);
    ssvhwa_write8(I2SR,0xA0);
    ssvhwa_write8(I2CR,0x00);
    ssvhwa_write8(I2CR, 0x90);          // enable I2C as master
    printf("init done \n");
    if (argc>1) duree=atoi(argv[1]); if (duree>0x3f) duree=0x3f;
    snd_cmd(0x12,0xa4);                 // 0x80|0x24 = soft reset
    snd_cmd(0x11,duree);     // select fps: 50=1 fps, max=0x3F=63
#ifdef color
 // select RGB
    snd_cmd(0x12,0x2C);
#endif
#ifdef huit                                     // select 8 bit mode
    snd_cmd(0x13,0x21);
#endif
    ssvhwa_close();return(0);
}
```

Notons cependant un certain nombre d'erreurs dans la *datasheet* de l'OV6620, notamment concernant le registre 0x11. Ce registre nous est utile pour abaisser la vitesse à laquelle la caméra capture et

transfert les images. Nos captures se faisant, dans un objectif de simplifier au maximum les aspects matériels de l'interfaçage, de façon totalement logicielle, il nous faut absolument abaisser cette vitesse pour atteindre au mieux 2 images par seconde. Pour ce faire, il nous faut écrire dans le registre 0x11 (facteur de division de l'horloge interne), fonction qui est documentée comme impossible dans la datasheet (registre 0x11 en lecture uniquement). Nous avons pu vérifier, tel que décrit dans [13], que le registre 0x11 est bien accessible en écriture. L'adresse de la caméra n'est pas non plus nécessairement documentée, nécessitant une exploration systématique des 127 adresses paires possibles (le dernier bit de parité étant utilisé dans le protocole I²C pour indiquer une lecture – valeur impaire – ou une écriture – valeur paire). Par exemple, alors que le capteur CMOS OV6620 que nous utilisons est documenté correctement comme répondant aux accès aux adresses 0xC0 et 0xC1, une exploration des adresses est nécessaire pour découvrir que la caméra 3 Mpixels OV3620 répond à l'adresse 0x60 (adresse paire donc en écriture, l'assignation des registres intéressants étant la même que pour l'OV6620).

### 5.3.2 Capture d'une image

D'un point de vue matériel, la seule contrainte de connexion d'une caméra OV6620 au Coldfire 5282 est l'abaissement des signaux 5 V de la caméra en une logique 3,3 V pour le processeur, opération effectuée par des *latchs* 74LV573 alimentés en 3,3 V (figure 4). Une contrainte supplémentaire provient de la rapidité des signaux lus : le transfert le plus rapide que peut supporter une lecture totalement logicielle des signaux issus de la caméra tel que décrit ici implique une période des signaux de pixels de 2,5 microsecondes (pour un taux de transfert de 2 images/seconde). Or nous avons constaté que la capacité électrique de certaines entrées – notamment les bits servant aussi de convertisseur analogique/numérique – est trop importante pour supporter des transitions aussi rapides. Nous précisons donc de connecter les latchs directement au bus de données du Coldfire 5282 (signaux D0-D7) et de n'imposer la valeur du bus (signal OE# des latchs) que lors de la lecture d'une adresse libre déclenchant un des signaux *Chip Select* (Csi# ; i=1..3 du Coldfire). Ce processeur a en effet l'avantage d'inclure en interne un décodeur d'adresse qui déclenche les signaux Csi# pour les conditions décrites dans le tableau 1.

| Signal | Plage d'adresses | broche SSV |
|---|---|---|
| CS1# | 0x1000.000-0x100F.FFFF | 48 |
| CS2# | 0x1010.000-0x101F.FFFF | 47 |
| CS3# | 0x1020.000-0x102F.FFFF | 46 |

***Tab. 1 : Tableau récapitulatif des adresses associées à chaque signal de Chip Select disponible en sortie du Coldfire 5282. Nous utilisons pour notre application CS2# et CS3#.***

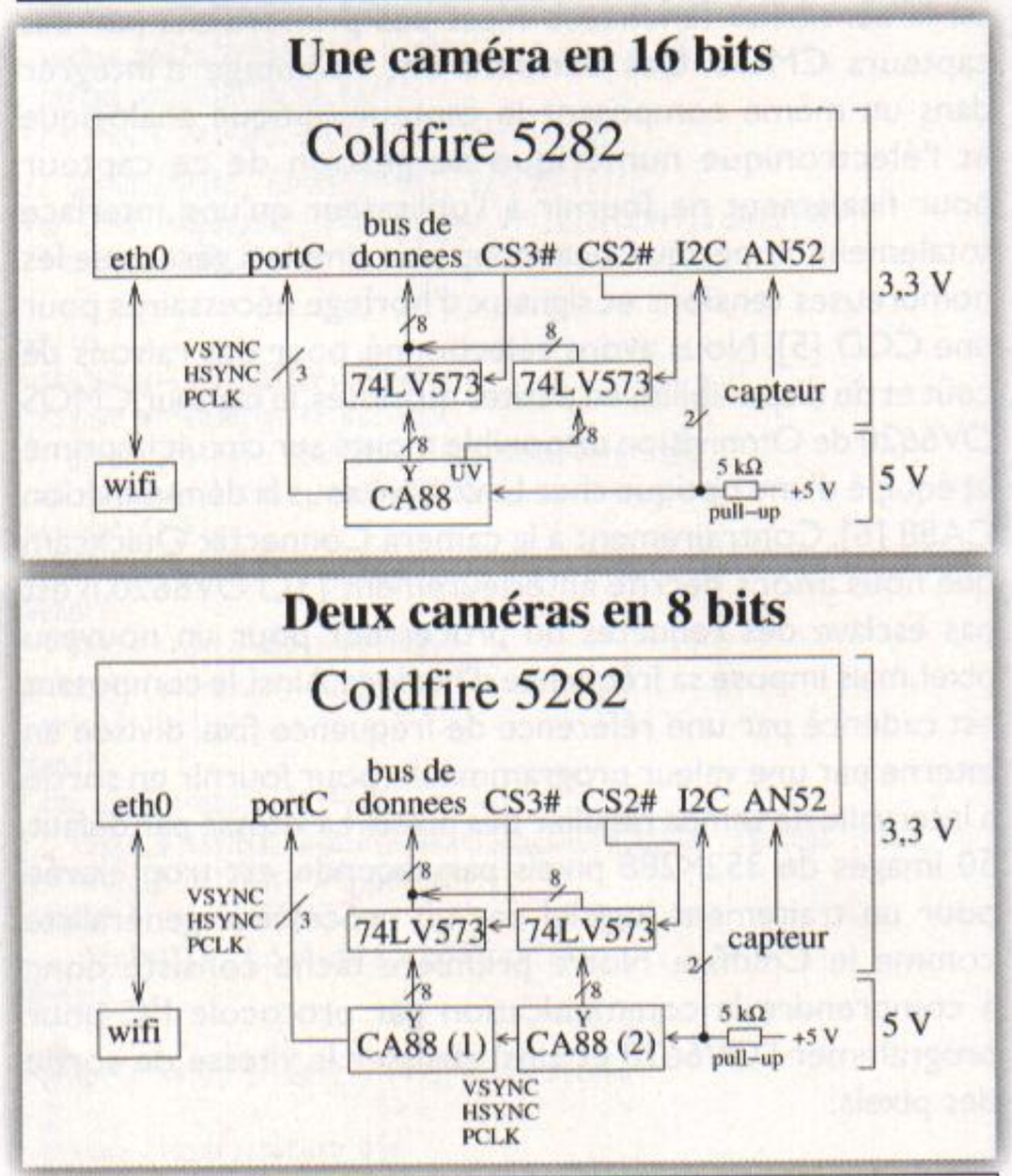



***Fig. 4 : Schéma de principe de la connexion des caméras CA88 au Coldfire 5282 avec les 74LV573 chargés de convertir la logique 5 V en 3,3 V requis par la carte embarquée, de la commande I²C des caméras et éventuellement d'une entrée analogique pour une mesure et transmission de paramètre.***

### 5.3.3 Capture d'une image couleur

Trois signaux sont analysés lors de la capture d'une image : VSYNC annonce par un passage au niveau bas le début d'une nouvelle image, HREF annonce tant qu'il est au niveau haut la validité des pixels le long d'une ligne (le passage au niveau bas de HREF est donc le signal de début d'une nouvelle ligne) et finalement PCLK annonce la validité (sur son front montant) d'une valeur de pixel. Les pixels sont soit transférés comme deux mots de 8 bits présentés simultanément sur les bus Y et UV de la caméra, ou comme un octet de 8 bits sur le port Y (UV étant alors inutilisé). Deux points importants sont à noter :

- PCLK continue à osciller pendant que HREF est au niveau bas. Le test de ce second signal pour valider un pixel est donc fondamental (il n'est pas suffisant de tester uniquement PCLK et négliger HREF)
- Le rapport cyclique du signal issu de PCLK est 50% en mode 16 bits (tel que annoncé dans la datasheet), mais en mode 8 bits ce signal n'est qu'un fin pulse d'une centaine de nanosecondes même en abaissant la fréquence de l'horloge interne après écriture dans le registre 0x11. Afin de transformer le signal issu de PCLK en un pulse suffisamment long pour être détecté

avec certitude de façon logicielle, un circuit additionnel est inséré entre la caméra et le Coldfire afin d'allonger PCLK à une valeur constante déterminée par la constante de temps du filtre RC présenté figure 5. Nous avons choisi un circuit à base de *trigger* de Schmitt car ce composant était disponible, mais de nombreuses autres solutions (par exemple à base de *timer* 555 monté en monostable) sont possibles.

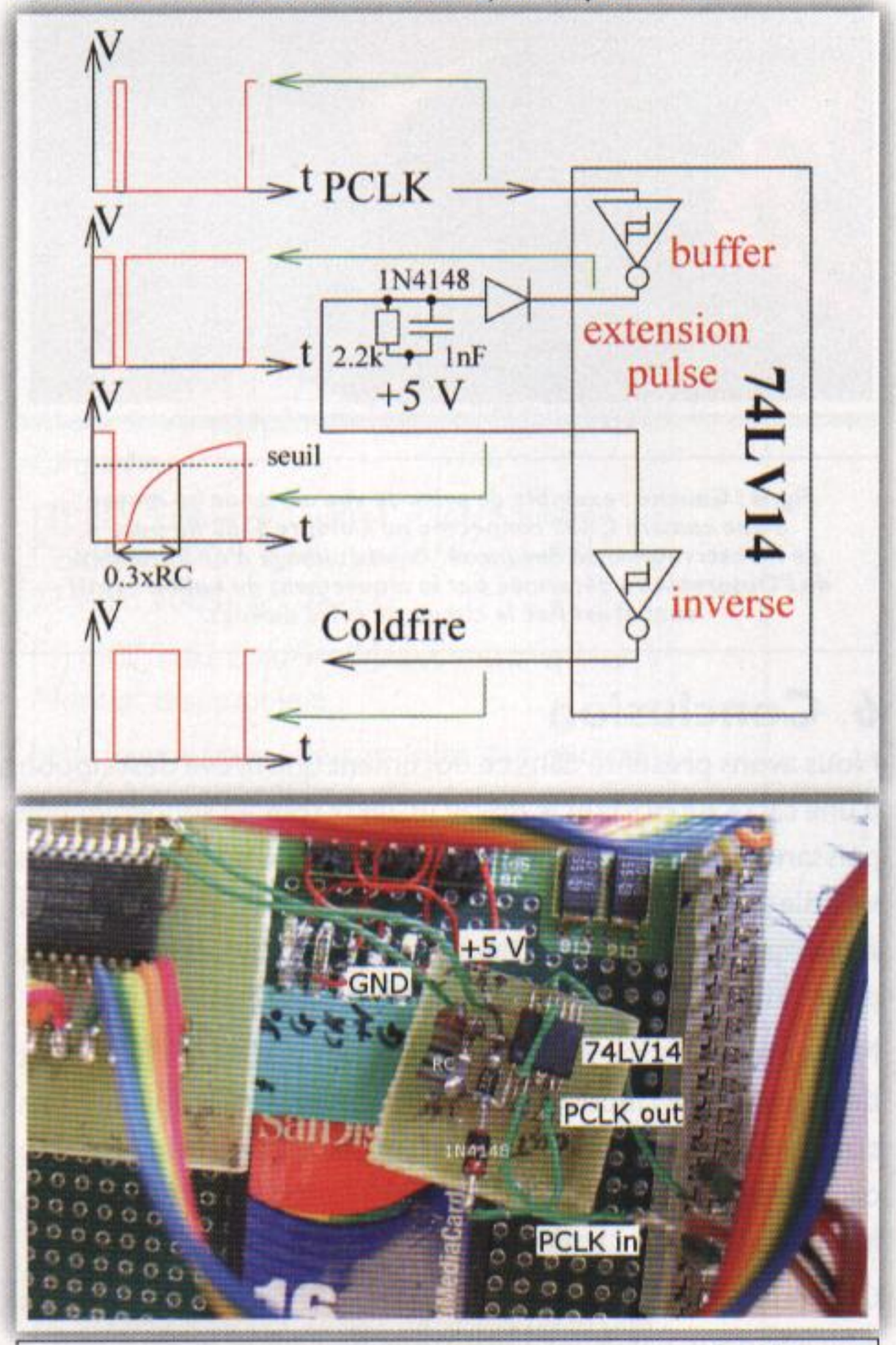



*Fig. 5 : Circuit pour l'élargissement des impulsions PCLK lors de l'utilisation de la caméra en mode 8 bits. Le premier inverseur sert de tampon dont la sortie charge le circuit RC au travers d'une diode. Après la fin du pulse court (<100 ns) qui a servi à décharger rapidement le condensateur, la diode empêche la recharge du condensateur par la sortie du trigger de Schmitt et la largeur de l'impulsion résultante → telle que détectée par le Coldfire est déterminée par la constante de temps du circuit RC : →≈ 0.3RC. Ce circuit est inutile en mode 16 bits où le rapport cyclique de PCLK est de 50% (tel que décrit dans la datasheet de l'OV6620) mais n'en affecte pas son fonctionnement : il peut donc rester connecté quel que soit le mode d'utilisation de la caméra.*

### 5.3.4 Organisation d'un capteur CMOS couleur

Une erreur de documentation concernant le mode 8 bits nous est apparue dans la datasheet de l'OV6620 que nous rectifions ici avec le schéma de la figure 6. En effet contrairement à ce qui semble être indiqué dans la documentation, la sortie en mode 8 bits est une alternance de valeurs RG et BG (et non une ligne complète de BG suivie d'une ligne complète de RG tel que nous interprétons la figure 4 (bas) de la datasheet de l'OV6620, version 1.4 datée du 13 Mai 2000). Nous vérifions la validité de notre analyse en convertissant les données brutes capturées sur les bus de la caméra (présentées ici comme une image en tons de gris) en des images couleur (format PPM) qui présentent bien ce qui est observé en réalité (notamment l'arc-en-ciel de la nappe de câble), tel que présenté sur la figure 7.

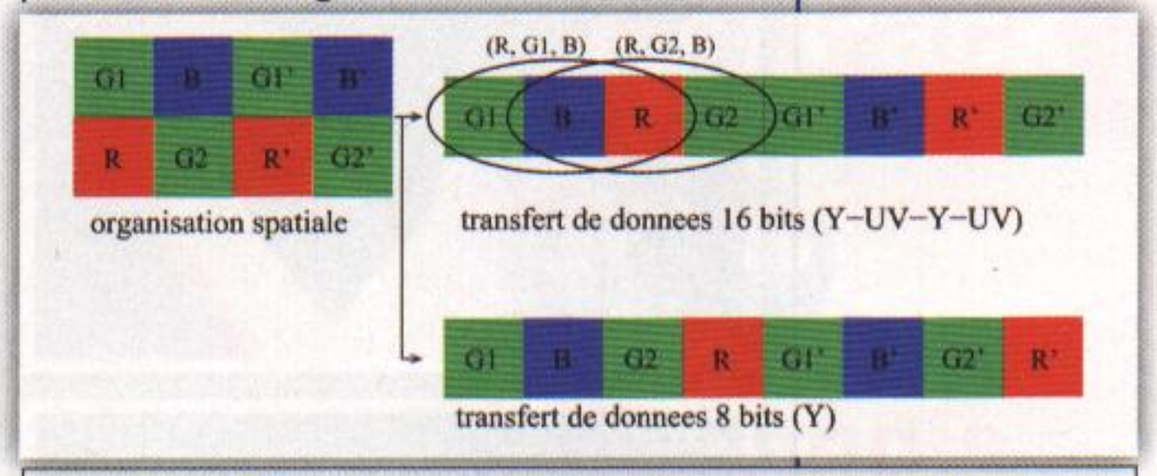


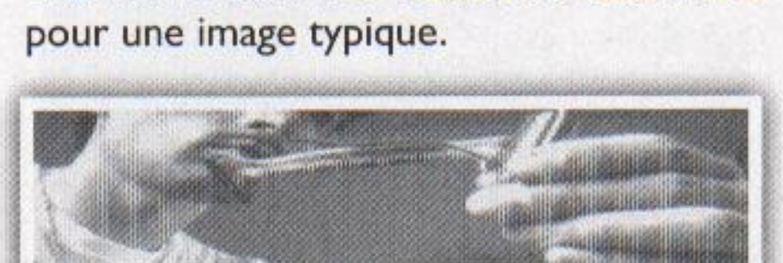

*Fig. 6 : Organisation des données telles que reçues du capteur CMOS OVL6620 ($G_1$, B, R, $G_2$ soit deux pixels verts pour chaque pixel rouge et bleu) et la conversion que nous avons choisi d'utiliser pour générer deux pixels (R, $G_1$, B) et (R, $G_2$, B) pour chaque quartet de données acquis.*

Nous avons implémenté un couple client/serveur où :

▶ Un module noyau capture les images et les transfère en espace utilisateur lors de la requête `read()`. Ce module noyau désactive les interruptions au cours d'une capture d'image (pendant environ une demi-seconde :fonctions `save_flags(flags);` dans un `unsigned long flags;` suivi de `cli();` – les interruptions sont réactivées en fin de capture d'une image par `restore_flags(flags);` qui implicitement appelle `sti();`) faute de quoi un certain nombre de pixels est perdu au cours d'une acquisition.

▶ Le programme utilisateur est dans ce cas un serveur *multithreadé* (librairie `pthread` fournie par uClinux).

▶ Plusieurs clients fonctionnant sous Unix se connectent au serveur, réceptionnent les images transférées par Ethernet (TCP/IP), convertissent les données brutes lues des caméras en images RGB pour affichage par les fonctions X11 (fonctions `XCreateSimpleWindow()` et `XPutImage()`). En convertissant les données brutes en image RGB au niveau du client, nous gagnons un facteur 2/3 en taille des paquets transférés (4 octets donnent 2 pixels RGB). Nous avons implémenté une compression sans perte (telle que fournie par la `libz` de uClinux) mais les résultats sont décevants : la compression est longue et ne permet de gagner qu'environ 15% sur la taille des données à transférer pour une image typique.



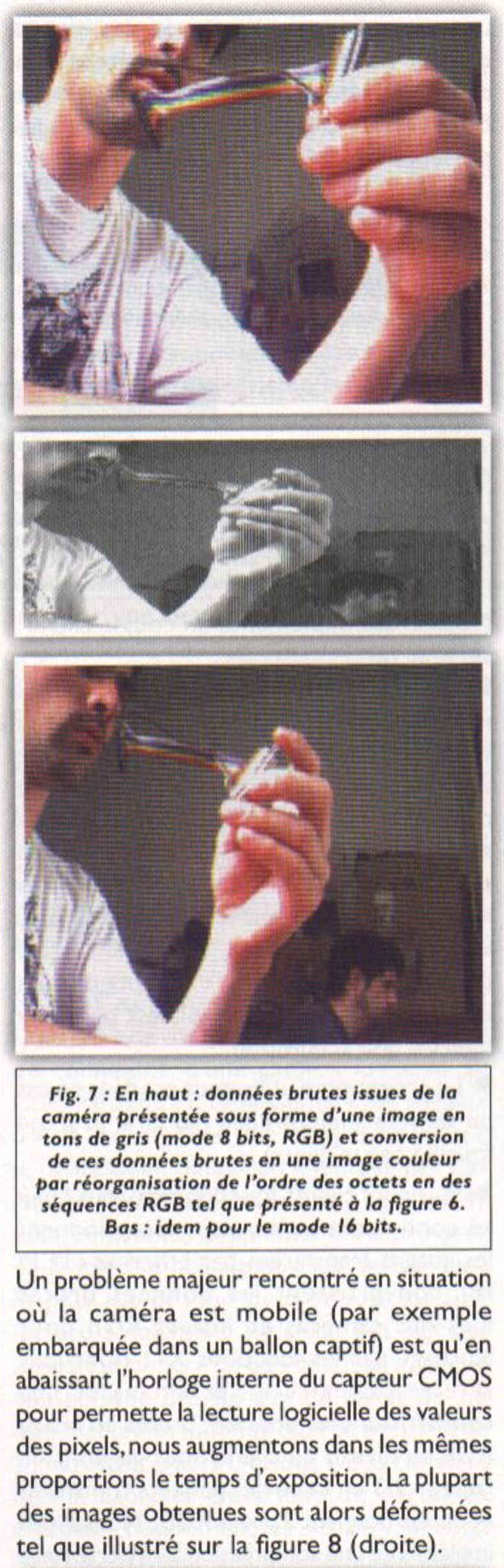


*Fig. 7 : En haut : données brutes issues de la caméra présentée sous forme d'une image en tons de gris (mode 8 bits, RGB) et conversion de ces données brutes en une image couleur par réorganisation de l'ordre des octets en des séquences RGB tel que présenté à la figure 6. Bas : idem pour le mode 16 bits.*

Un problème majeur rencontré en situation où la caméra est mobile (par exemple embarquée dans un ballon captif) est qu'en abaissant l'horloge interne du capteur CMOS pour permette la lecture logicielle des valeurs des pixels, nous augmentons dans les mêmes proportions le temps d'exposition. La plupart des images obtenues sont alors déformées tel que illustré sur la figure 8 (droite).

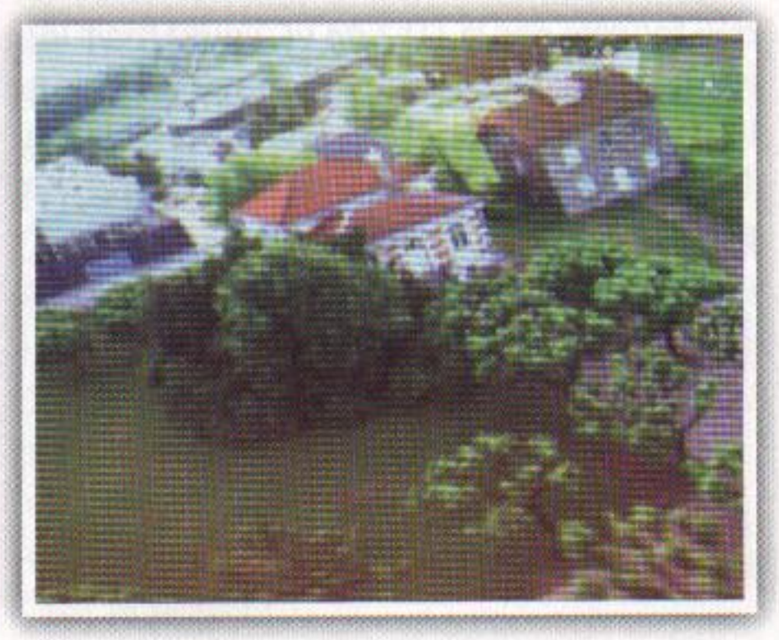


*Fig. 8 : Gauche : exemple de prise de vue aérienne au moyen d'une caméra CA88 connectée au Coldfire 5282 du parc de l'Observatoire de Besançon. Droite : image d'un bâtiment de l'Observatoire déformée par le mouvement du ballon captif auquel est fixé le circuit de prise de vues.*

## 6 Conclusion

Nous avons présenté dans ce document une brève description d'une carte supportant le noyau uClinux tout en apportant une puissance de calcul importante, une quantité de mémoire non volatile et volatile suffisante pour la plupart des applications embarquées et une large panoplie de ports de communication, pour un prix de l'ordre de la centaine d'euros.

Nous avons illustré le mode de programmation de la plupart de ces périphériques par la mise en œuvre d'applications concrètes : stockage sur carte MultiMediaCard, interfaçage de capteurs sur les ports de conversion analogique/numérique, interfaçage d'une caméra couleur.

Ces résultats ont nécessité un certain nombre de développements logiciels pour palier les limitations d'uClinux et s'adapter à cet environnement de développement : modules noyaux pour accéder aux registres contrôlant le matériel fourni avec le processeur Coldfire 5282, méthodes efficaces de gestion et d'occupation de la mémoire. Nous présenterons ultérieurement quelques applications concrètes d'instruments embarqués basées sur ce matériel.

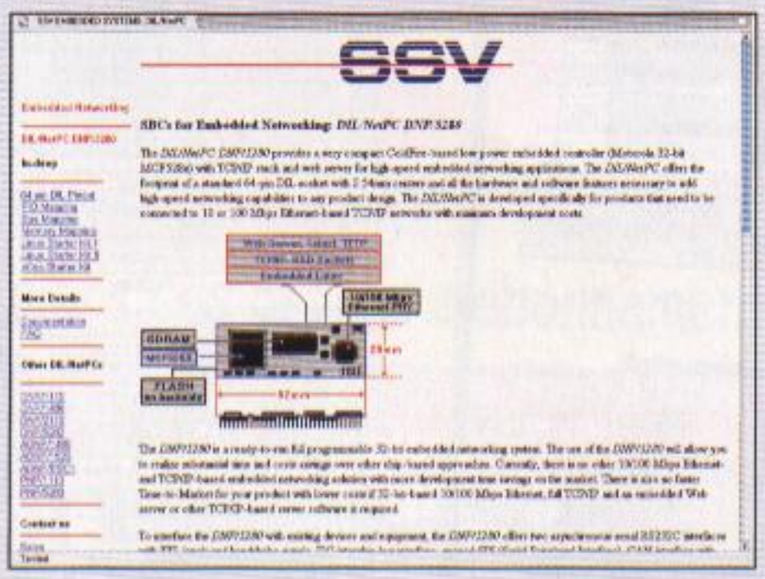
## Remerciements

Nous remercions F.Vernotte, directeur de l'Observatoire de Besançon, pour son hébergement gracieux de l'association étudiante Projet Aurore dans ses locaux, ainsi que Christian Ferrandez (FEMTO-ST/LPMO, Besançon) pour le prêt de la carte SSV et de l'OV3620.

Vincent Giordanno et l'équipe « métrologie des oscillateurs » du laboratoire FEMTO-ST/LPMO nous ont fourni l'hélium pour la ballon captif. L'association de diffusion des Logiciels libres sur la Franche-Comté – Sequanux (**www.sequanux.org**) – est remerciée pour son support logistique.

## LIENS

[1] http://www.dilnetpc.com/dnp0038.htm




[2] http://www.freescale.com/webapp/sps/site/overview.jsp?nodeId=0162468rH3YTLCFqnN


[3] J.-M Friedt, S. Guinot, « Introduction au Coldfire 5272 », GNU/Linux Magazine France 73 (Juin 2005), pp. 26-33.

[4] MCF5282 *ColdFire Microcontroller User's Manual*, disponible à :

http://www.freescale.com/files/dsp/doc/ref_manual/MCF5282UM.pdf

[5] R. Berry, V. Kanto, J. Munger, *The CCD Camera Cookbook*, Willmann-Bell

[6] http://www.lextronic.fr/Capteurs/CAMERA.htm

[7] http://friedtj.free.fr/mmc_pport.pdf

[8] http://friedtj.free.fr/aduc816.pdf

[9] SanDisk MultiMediaCard & Motorola 8 bits microcontroller Interface Design Reference Example Rev. 2 (04/1999), Sandisk.

[10] Datasheet HB288016MM1 MultiMediaCard 16 MByte Rev 0.1 (Nov. 24, 1999), Hitachi, ou HB28E016MM2 disponible à http://www.ulrichradig.de/site/atmel/avr_mmcsd/pdf/hitachi_hb28b128mm2.pdf

[11] http://www.ulrichradig.de/site/atmel/avr_mmcsd/pdf/MMCSDTimming.pdf

[12] D. McCullough, Why is Malloc Different Under uClinux ?, disponible à http://www.linuxdevices.com/articles/AT7777470166.html

[13] I.N. Oiza, Digital Camera Interface (Mai 2004), disponible à http://www.robozes.com/inaki/dproject



**Jean-Michel Friedt,**

**friedt@free.fr**

**Simon Guinot,**

**simon@sequanux.org**

**Association Projet Aurore, UFR-ST La Bouloie**
**16 route de Gray, 25044 Besançon**