GNU LINUX MAGAZINE / FRANCE

L 19275 - 92 - F: 6,20 €

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

▶MARS ▶2007 ▶NUMÉRO 92

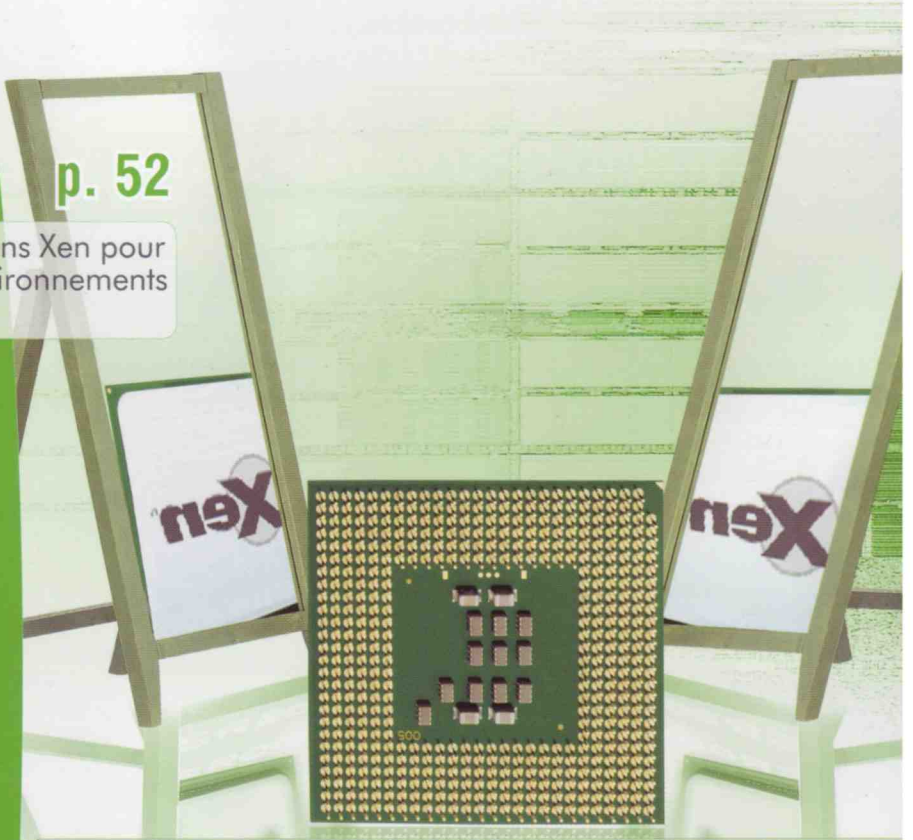
Paravirtualisation

Xen & SLO

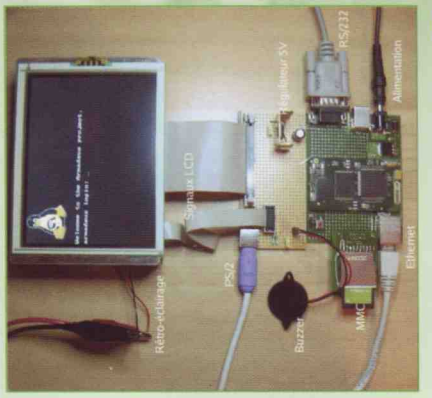
Répartition de charge

p. 52

Découvrez les mécanismes à votre disposition dans Xen pour gérer les priorités applicatives entre plusieurs environnements partagés au sein d'un même serveur.



Armadeus : l'embarqué ARM accessible à tous !



▶ARM9, 16Mo SDRAM, 8Mo Flash NOR, FPGA, CAN ...

Faites connaissance avec le projet associatif Armadeus au travers d'une utilisation concrète de sa carte de développement construite autour d'un Freescale i.MXl, un ARM920T 192 Mhz.

p. 70

Administration et développement sur systèmes UNIX

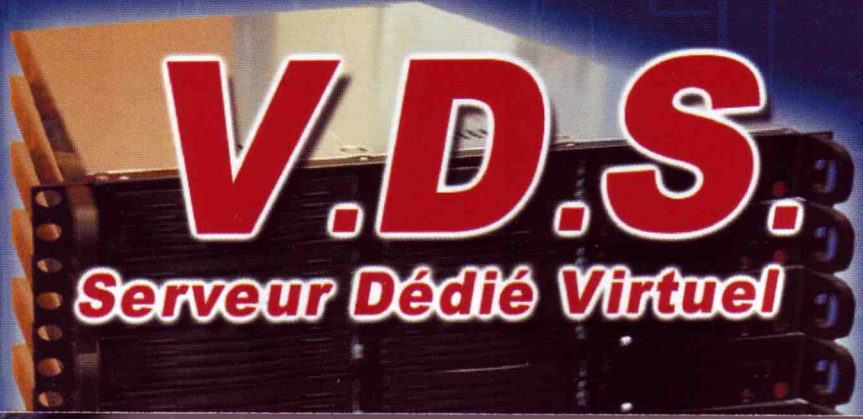
V.D.S.

- *Un système dédié complet sans aucune limitation. (Distribution Debian accès root intégral - panel de gestion ultra simplifié).*
- *Des ressources réservées et confortables (Ram, CPU, I/O, Bande Passante).*
- *Un prix bas pour un rapport qualité/prix unique.*

Bande passante incluse de 2 Mbps à 6 Mbps en trafic illimité - Espace disque de 4 à 40 Go - mémoire vive de 64 Mo à 512 Mo

à partir de **9 €** HT/mois

www.sivit.fr



Sivit

Edito

C'est le printemps !

Enfin, pas vraiment, certes, tout dépend de la date à laquelle vous lisez ce magazine. Quoi qu'il en soit, ce numéro a déjà une belle couleur verte de saison. Vous ne le saviez peut-être pas, mais la couleur de la couverture n'est pas qu'une question de contraste avec le précédent numéro et nos autres publications.

Et puis le vert, c'est aussi l'opposé du rouge. Rouge comme la couverture du dernier hors-série GLMF consacré aux *BSD.

« Quoi ? Que dit-il ? Que lis-je là ? »

Eh oui, non seulement j'ose le faire, mais en plus j'ose le dire ici-même : il existe d'autres systèmes d'exploitation en Logiciel libre et certains disposent de fonctionnalités qu'il est plaisant d'utiliser et qui n'existent pas sous cette forme dans GNU/Linux (notez que je ne m'avance pas trop non plus).

« Damn ! Le voici qui revendique ses méfaits !»

Il faut savoir s'ouvrir à de nouveaux horizons (tant que ceux-ci restent « propres »). L'objectif de GLMF et son contenu sont donnés en couverture : Administration et développement sur systèmes UNIX. Et puis, après tout, nous avions déjà parlé de GNU/Hurd et OpenSolaris...

« Mais... »

Tsss, Tsss... ! Il suffit. À chaque *Solutions Linux/Linux Expo*, un utilisateur BSD vient me voir pour me demander pourquoi nous n'avons jamais d'article sur son système. Chaque année, je réponds que je n'ai rien contre l'idée, qu'il m'en écrive un et nous verrons bien s'il peut être publié. Chaque année, la réponse est la même... « Ah non, pas moi ! ». Mais cette année-ci, la question ne s'est pas posée :)

Sur ce, je vous laisse joyeusement (c'est le printemps, je l'ai dit plus haut) découvrir ce numéro dont je suis particulièrement satisfait et vous donne rendez-vous le 24 mars prochain pour la suite...

Denis Bodor

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

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

Impression :
VPM Druck Allemagne
Distribution France :
(uniquement pour les dépositaires de presse)
MLP Réassort :
Plate-forme de Saint-Barthélemy-d'Anjou.
Tél. : 02 41 27 53 12
Plate-forme de
Saint-Quentin-Fallavier.
Tél. : 04 74 82 63 04
Service des ventes :
Distri-médias :
Tél. : 05 61 72 76 24

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

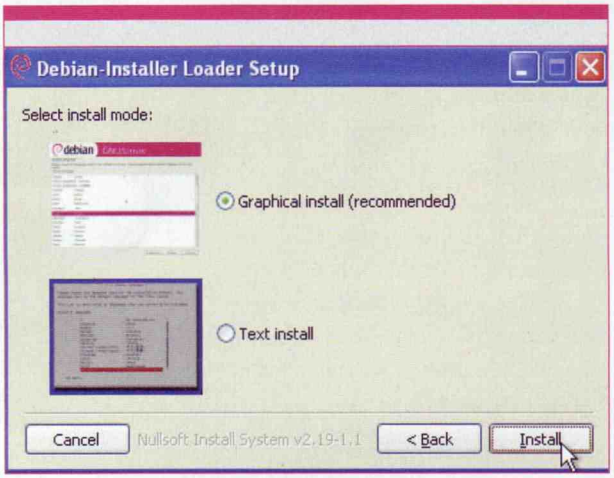
WWW.GNULINUXMAG.COM

Denis Bodor

Goodbye Microsoft !

En voilà un titre accrocheur pour un si petit article ! Propagande ? Non, il s'agit tout simplement d'un site vitrine proposant d'installer une distribution Debian GNU/Linux... depuis Windows.

Après l'installation classique via CD *bootable*, les méthodes basées sur `deboostrap`, l'installation via le Web et l'installation à distance « *over* SSH », voici l'installeur Debian sous forme d'exécutable Windows. Comme le montrent les captures présentées sur cette page, il s'agit d'un programme tout ce qu'il y a de plus visuellement Windows.

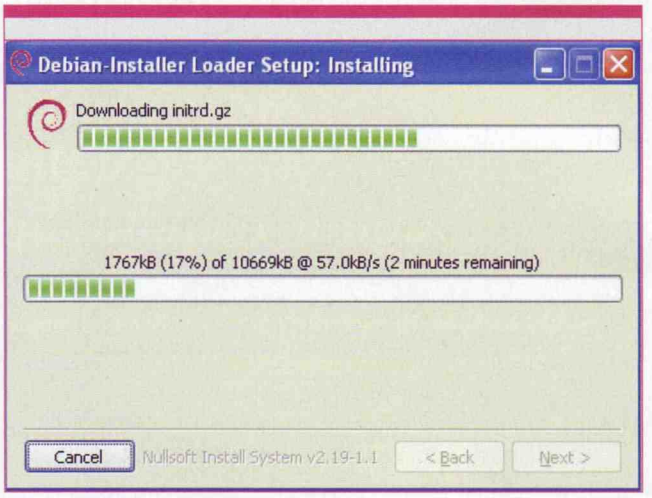


En réalité, Robert Millan a simplement créé un chargeur d'install Debian. Son application télécharge en effet une image Netboot de l'installeur Debian (une image du noyau Linux et une image `initrd/initramfs`), puis utilise la partie cliente DOS de GRUB (`grub4dos`) pour préconfigurer le prochain redémarrage de la machine.

À ce moment, la machine offrira un choix entre le démarrage sur l'installation de Windows préexistante et l'installeur Debian. Notez que le programme Windows détermine automatiquement la configuration 32 ou 64 bits.

L'idée n'est pas originale, un programme équivalent existe pour la distribution Ubuntu et est disponible au stade de prototype (**https://wiki.ubuntu.com/install.exe**).

Les sources du programme Debian sont bien évidemment disponibles en GPL. Elles utilisent une version Debian du système d'installation scriptable de NullSoft. Vous les trouverez sur le serveur SVN Debian (`svn co svn://svn.debian.org/d-i/people/rmh/win32-loader`). La construction se fait très simplement sous GNU/Linux grâce à l'installation des paquets `nsis`, `mingw32`, `iso-codes` et `gettext`.



Il reste encore pas mal de travail à faire sur cette application. Toutes les versions de Windows ne sont pas supportées et aucune détection de version n'est effectuée. D'autre part, la gestion des erreurs reste à améliorer.

L'objectif principal du projet est de fournir une méthode d'installation encore plus simple qu'un CD bootable à l'utilisateur lambda. Bien entendu, elle reste très intéressante dans d'autres situations comme l'installation rapide d'un GNU/Linux sur une machine Windows en entreprise.

Denis Bodor,

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

LIENS

- Goodbye Microsoft : **http://goodbye-microsoft.com/more.html**
- GRUB4DOS/WINGRUB : **http://grub4dos.sourceforge.net/** **http://grub4dos.freespaces.com/**

La qualité à votre portée

Serveurs Dédiés IBM x Séries

IDC avec toutes les garanties de sécurité

Support technique gratuit par mail et téléphone

Garantie 30 jours Satisfait ou Remboursé

Adresses Ips illimitées

Devis en ligne et sur mesure

...

Découvrez tous les avantages de travailler avec la meilleure marque. Arsys, hébergeur professionnel depuis plus de 10 ans, vous offre votre propre serveur IBM ou Supermicro avec toute la technologie, sécurité et qualité de service à partir de **99 €**/mois.

Découvrez avec arsys.fr l'internet de qualité.

arsys.fr
internet de qualité

Noms de Domaine | Hébergement | Serveurs Dédiés | Applications

Dédié Générique
Dédié Administré
Dédié de Courrier

www.arsys.fr / 0800 940 865
Appel Gratuit

Éric Lacombe, Matthieu Barthélemy
Auteur invité : Frédéric Raynal

Kevent, kpage et stabilité vs pilotes

Pour cette nouvelle édition du Kernel Corner, nous vous proposons trois brèves dont la deuxième est écrite par Frédéric Raynal (rédac chef de MISC). Il nous présente un module qu'il a développé à des fins pédagogiques pour découvrir le fonctionnement de la pagination. La dernière brève développe une analyse sur les problèmes liés aux crashs des drivers affectant la stabilité du système et les solutions proposées et disponibles pour y faire face. Pour la première brève, le fonctionnement de l'infrastructure de gestion unifiée des événements, Kevent, est expliqué.

KEVENT : MÉCANISME GÉNÉRIQUE DE GESTION DES ÉVÉNEMENTS

Éric Lacombe – tuxiko@free.fr – eric.lacombe@security-labs.org

Introduction

Kevent est un mécanisme générique de gestion d'événements créé par Evgeniy Polyakov. Au travers d'une interface unifiée permettant la gestion de toutes sortes d'événements (*socket*, *poll/select*, *timer*, signal, etc.), il permet aux programmes utilisateurs y faisant appel, d'être averti lors de leur survenance. Il a été conçu dans le même esprit que les *completion ports* de MS Windows ou que l'infrastructure *kqueue* de FreeBSD/OS X. En utilisant un seul appel système, un *thread* peut récupérer tous les types d'événements que le noyau est capable de générer. Cela, à la place des anciennes interfaces qui n'autorisent que la réception de types spécifiques d'événements (par exemple : l'expiration de timers, l'arrivée d'un nouveau message dans une file, etc.).

Son utilisation passe par la définition de nouveaux appels système. Le projet en est, lors de l'écriture de ces lignes, à sa 35ème révision ! Avant d'être incluse dans la *mainline*, il est nécessaire que l'API soit irréprochable, car alors il ne sera plus possible de la changer (étant liée à l'espace utilisateur). Ulrich Drepper, mainteneur de la glibc, a participé activement à la critique de l'API et des fonctionnalités. Ainsi, la version actuelle est très proche de la maturité nécessaire à l'inclusion dans la mainline.

Kevent supporte la définition d'événements déclenchés par front ou par niveau. Dans certaines situations, il est semblable à *poll/epoll* tout en étant plus rapide et supportant bien mieux le passage à l'échelle. Il a été conçu pour travailler avec quasiment n'importe quel type d'événement (par exemple les signaux POSIX ;).

Descripteur d'un événement : structure ukevent

Tout d'abord jetons un œil au descripteur d'un événement au niveau utilisateur. Cette structure va servir lors de l'ajout, de la modification ou de la suppression d'une requête au noyau sur la notification d'un type d'événement, mais également pour signifier qu'un événement a été consommé (c.-à-d. utilisé) par l'application.

```
struct ukevent
{
  /* Identifiant de cette requête, ex : numéro
     de socket, descripteur de fichier, etc. */
  struct kevent_id        id;
  /* Type de l'événement, ex : KEVENT_SOCK,
     KEVENT_INODE, KEVENT_TIMER, etc. */
  __u32                   type;
  /* L'événement lui-même, ex : SOCK_ACCEPT,
     INODE_CREATED, TIMER_FIRED, etc. */
  __u32                   event;
  /* Flags propre à chaque événement et définis
     lors de la requête */
  __u32                   req_flags;
  /* Flags propre à chaque événement et défini
     par le noyau
   * ex : KEVENT_REQ_ONESHOT, KEVENT_REQ_READY, etc. */
  __u32                   ret_flags;
   /* Données retournées par l'événement. Au choix du
     responsable du déclenchement de l'événement. */
  __u32                   ret_data[2];
   /* Données de l'utilisateur, non utilisée, juste
     copiées depuis et vers l'espace utilisateur.
   */
  union {
          __u32           user[2];
          void            *ptr;
        };
};
```

Nous ne rentrerons pas, par la suite, dans le détail de tous les membres de cette structure. Nous expliquerons l'infrastructure Kevent vue depuis l'espace utilisateur et, ensuite, nous relierons les éléments clés aux activités correspondantes se déroulant au sein du noyau. Cette dernière étape est à voir comme une introduction pour le lecteur souhaitant approfondir le sujet.

API de Kevent

L'interaction avec le noyau pour effectuer un ajout, une modification, une suppression de requête sur événement, ou bien signifier qu'un événement est prêt

KEVENT : MÉCANISME GÉNÉRIQUE DE GESTION DES ÉVÉNEMENTS

Éric Lacombe – tuxiko@free.fr – eric.lacombe@security-labs.org

(ce qui permet de réveiller un ou plusieurs threads en attente sur cet événement), passe par l'utilisation de la fonction `kevent_ctl`. Avant d'appeler cette dernière, il est nécessaire de demander au noyau de créer une structure pour cataloguer les futurs événements que nous souhaitons surveiller. Pour cela, la première méthode est d'ouvrir le fichier `/dev/kevent`. Ainsi, un descripteur propre à l'application (dont l'élément principal est une liste) est créé par le noyau et un identifiant permettant le dialogue avec le noyau est retourné par l'appel système `open`. Une autre méthode est également proposée pour répondre à certaines situations que nous allons bientôt expliquer.

```
int kevent_ctl(int fd,
               unsigned int cmd,
               unsigned int num,
               struct ukevent *arg)
```

Cet appel système est celui qui nous permet de gérer nos requêtes de notification sur événements. Ces paramètres sont les suivants :

- `fd` : Il s'agit du descripteur de fichier (l'identifiant), correspondant à la file d'événement Kevent utilisée par l'application.
- `cmd` : Il s'agit de la commande à exécuter. Les choix possibles sont : `KEVENT_CTL_ADD`, `KEVENT_CTL_REMOVE`, `KEVENT_CTL_MODIFY`, `KEVENT_CTL_READY` pour respectivement, ajouter, supprimer, modifier une requête de notification ou marquer une notification comme étant prête afin de réveiller les threads l'attendant.
- `num` : Il s'agit du nombre de requêtes qui sont soumises lors de l'appel.
- `arg` : Nous fournissont dans cet argument un pointeur sur les structures `ukevent`, définissant nos requêtes.

Après avoir ajouté des requêtes de notification sur certains types d'événements via l'utilisation de `kevent_ctl`, le noyau va récupérer les événements émis par les sous-systèmes correspondants et les placer dans une structure `struct kevent_user` accessible seulement depuis l'espace noyau et définie pour chaque utilisateur. Nous revenons par la suite sur cette structure.

Afin de récupérer des événements depuis l'espace utilisateur, deux solutions sont proposées. La première est une version synchrone qui va donc déclencher l'écriture des événements qui se produisent lors de l'appel. Dans la seconde version, qui est asynchrone, l'utilisateur va récupérer les événements au travers d'un buffer qui est rempli progressivement par le noyau. Cette dernière méthode est préconisée pour les applications nécessitant des performances optimales.

Interface synchrone

L'interface synchrone est la plus simple à utiliser, mais aussi la moins performante. Elle passe par l'utilisation de la fonction suivante retournant le nombre d'événements copiés et qui peut être utilisée de façon bloquante ou non via le paramètre `timeout` :

```
int kevent_get_events(int ctl_fd,
                      unsigned int min_nr,
                      unsigned int max_nr,
                      struct timespec timeout,
                      struct ukevent *buf,
                      unsigned flags)
```

- `ctl_fd` : Comme précédemment, il s'agit de l'identifiant du descripteur de la file Kevent.
- `min_nr` : Le nombre minimal d'événements à renvoyer pour que la fonction retourne à l'appelant.
- `max_nr` : Le nombre de structures `ukevent` que nous fournissons via le tampon mémoire `buf`
- `timeout` : Le temps d'attente avant de retourner à l'appelant si moins de `min_nr` événements ont été produits.
- `buf` : Le pointeur sur la table des `ukevent`
- `flags` : Ce drapeau peut prendre actuellement seulement la valeur `KEVENT_FLAGS_ABSTIME` pour signifier au noyau que la valeur de `timeout` est une valeur de temps absolue. Par défaut (flags à NULL), le `timeout` est une valeur définie relativement à la date actuelle.

Interface asynchrone

L'interface asynchrone est un peu plus complexe à mettre en place au niveau utilisateur. En effet, elle passe par l'initialisation d'un buffer au niveau de l'espace utilisateur dans lequel le noyau copie les événements produits depuis sa propre structure interne (dans l'espace noyau), en faisant attention à ne pas écraser les événements non encore consommés. Ainsi, l'application doit appeler la fonction `kevent_commit` afin de signifier au noyau où en est l'état de son buffer.

Le descripteur de ce buffer, une structure `kevent_ring`, est défini comme ci-dessous. Il s'agit d'une table circulaire (c.-à-d. qu'une fois arrivé à la fin de la table, la première entrée est de nouveau utilisée) dont la taille est au choix de l'application. Le membre `ring_kidx` correspond à l'index dans la table utilisée par le noyau pour y écrire un prochain événement. Le membre `ring_over` sert, quant à lui, à conserver le nombre de tours déjà effectués dans la table circulaire, afin que le noyau et l'application soient en phase sur l'état du buffer, avant que cette dernière ne marque des événements comme étant consommés.

```
struct kevent_ring
{
  unsigned int ring_kidx, ring_over;
  struct ukevent event[0];
}
```

Après la déclaration d'une telle structure, la fonction suivante est appelée par l'application. Elle fournit en retour un identifiant unique, correspondant à

KEVENT : MÉCANISME GÉNÉRIQUE DE GESTION DES ÉVÉNEMENTS

Éric Lacombe – tuxiko@free.fr – eric.lacombe@security-labs.org

la structure interne du noyau y étant associée (la structure `struct kevent_user`) et nécessaire aux futures interactions entre l'utilisateur et le noyau. Cet identifiant a exactement le même rôle que celui fourni lors de l'ouverture de `/dev/open`.

```
int kevent_init(struct kevent_ring *ring,
                unsigned int ring_size,
                unsigned int flags);
```

- `ring` : Pointeur sur le tampon circulaire ;
- `ring_size` : Taille du tampon circulaire en nombre d'événements ;
- `flags` : Comme toutes les fonctions précédentes, seul le drapeau `KEVENT_FLAGS_ABSTIME` est reconnu.

Pour récupérer à présent les événements, nous utilisons la fonction suivante qui retourne le nombre d'événements copiés dans le tampon circulaire :

```
int kevent_wait(int ctl_fd,
                unsigned int num,
                struct timespec timeout,
                unsigned int flags)
```

- `ctl_fd` : Comme précédemment, il s'agit de l'identifiant du descripteur de la file Kevent ;
- `num` : Nombre d'événements maximal à écrire dans le tampon circulaire ;
- `timeout` : Le temps d'attente maximal avant de retourner à l'appelant, pour que de la place se libère dans la file `kevent` ;
- `flags` : Comme toutes les fonctions précédentes, seul le drapeau `KEVENT_FLAGS_ABSTIME` est reconnu.

Pour renseigner le noyau sur le fait qu'un événement a été consommé, l'utilisateur appelle la fonction suivante qui retourne le nombre de `kevent` enregistrés comme étant consommés :

```
int kevent_commit(int ctl_fd,
                  unsigned int new_uidx,
                  unsigned int over);
```

- `ctl_fd` : Comme précédemment, il s'agit de l'identifiant du descripteur de la file Kevent ;
- `new_uidx` : L'index relatif à l'événement consommé ;
- `over` : Compteur de cycle dans le tampon circulaire pour la valeur `new_uidx` donnée (correspond au membre `ring_over` de la structure `kevent_ring`).

Dans le noyau

Pour chaque utilisateur de Kevent, le noyau initialise une structure du type `kevent_user`. Cette structure embarque un arbre binaire auto-équilibré de type `red black tree` pour le stockage des événements (le membre `kevent_root` pointe sur sa racine), ainsi qu'une liste doublement chaînée (cf. KC 86) pour accéder facilement aux événements qui sont prêts (via le membre `ready_list`). Notons que la file d'attente `wait` dans la structure est utilisée pour stocker temporairement les processus en attente sur un événement en dehors des listes de l'ordonnanceur. Finalement, remarquons que le tampon circulaire de l'espace utilisateur est disponible via le pointeur `pring` (la macro `__user` est utilisée, car l'adresse est à interpréter dans l'espace d'adressage virtuel propre à l'application et non celui du noyau).

```
struct kevent_user
{
  struct rb_root        kevent_root;
  spinlock_t            kevent_lock;
  /* Nombre de kevent enregistrés */
  unsigned int          kevent_num;
  /* Liste des kevents prêt */
  struct list_head      ready_list;
  /* Nombre de kevents prêt */
  unsigned int          ready_num;
  /* Utilisé pour la protection de la liste
     ready_list (cf. KC 87 sur les spinlocks) */
  spinlock_t            ready_lock;
  /* Mutex pour se protéger des manipulations
     simultanées de la structure kevent_user */
  struct mutex          ctl_mutex;
  /* File d'attente jusqu'à ce que les événements
     soient prêts */
  wait_queue_head_t     wait;
  int                   need_exit;
  /* Compteur incrémenté pour chaque nouveau kevent */
  atomic_t              refcnt;
  /* Mutex pour la protection sur l'accès depuis le
     noyau au tampon circulaire qui est en espace
     utilisateur */
  struct mutex          ring_lock;
  /* Index, taille, etc. */
  unsigned int    kidx, uidx, ring_size, ring_over, full;
  /* Pointeur vers le tampon circulaire
     en espace utilisateur */
  struct kevent_ring __user *pring;
  /* Utilisé pour les échéances donné avec une date
     et non une valeur relative */
  struct hrtimer        timer;
  /* Utilisé uniquement pour des événements privés de
     l'espace utilisateur s'ils ont été définis */
  struct kevent_storage  st;
};
```

Les événements stockés dans la structure précédente ne sont pas des structures `struct ukevent`. En fait, la représentation interne d'un événement au sein du noyau est une structure `struct kevent` contenant le `ukevent` correspondant à la requête émise par l'utilisateur.

```
struct kevent
{
  /* RCU Utilisé pour la libération de la structure
   * (cf. KC 89 pour une introduction sur ce moyen
   * de synchronisation) */
  struct rcu_head       rcu_head;
  struct ukevent        event;
  /* Pour la protection sur la modification de la structure */
```

KEVENT : MÉCANISME GÉNÉRIQUE DE GESTION DES ÉVÉNEMENTS

Éric Lacombe – tuxiko@free.fr – eric.lacombe@security-labs.org

```
    spinlock_t              ulock;
    /* Entrée dans le red black tree du kevent_user */
    struct rb_node          kevent_node;
    struct list_head        storage_entry;
    struct list_head        ready_entry;
    u32                     flags;
    /* Lien sur la structure propre à l'utilisateur */
    struct kevent_user      *user;
        struct kevent_storage   *st;
        struct kevent_callbacks callbacks;
    /* Données privées pour différent type de stockage.
     * Au choix du sous-système relatif à l'événement
     * concerné.
     */
    void                    *priv;
};
```

Notons que les membres `storage_entry` et `ready_entry` ne sont pas employés ensembles. En effet, s'il s'agit d'un événement classique, le membre `ready_entry` est utilisé pour la liaison avec la liste correspondante (`ready_list`) présente dans la structure `kevent_user` ; sinon, il s'agit d'un événement privé et l'autre membre est utilisé, auquel cas des données privées peuvent être utilisées par le sous-système via le membre `priv`.

Finissons en mentionnant le membre `callbacks` qui est une structure contenant trois pointeurs sur des fonctions. Chacune d'elles sert dans une situation bien précise. Une première est exécutée lorsque l'événement correspondant se produit, une autre lors du chaînage de l'événement (c.-à-d. de la structure) et la dernière lorsque la structure est enlevée de la file.

Sous-systèmes du noyau convertis à Kevent

Dans la version 35 du patch, huit sous-systèmes ont été convertis à l'utilisation de Kevent. Au niveau des sockets, Kevent permet de déclencher rapidement les notifications pour chaque commande `send`, `recv` ou `accept`, pour une socket donnée. Aussi, les méthodes poll/select de tous les drivers peuvent être utilisées au travers de Kevent. De même, au niveau des *pipes*, la notification peut s'effectuer sur la survenance des événements `send/recv` ou `pipe/fifo`. Les timers ne sont pas oubliés, et Kevent rend compte également de l'utilisation des *high-resolution timers*. Notons également le support de l'infrastructure des signaux par Kevent, rendant possible l'envoi des signaux au travers de la file d'événements de Kevent. Dans le même esprit, l'événement d'expiration des timers POSIX peut être véhiculé au travers de la file Kevent. Une autre fonctionnalité permet la définition de n'importe quel événement privé au niveau utilisateur et ensuite de le valider au moyen de la commande `kevent(KEVENT_READY)`.

Finalement, le dernier apport à Kevent a été l'implémentation de primitives asynchrones d'entrées/sorties (AIO) supportant l'infrastructure de Kevent. Ainsi les appels système `aio_sendfile` et `aio_sendfile_path` ont été définis et apportent un gain en performance significatif face à l'utilisation de `sendfile` qui est synchrone.

Conclusion

Pour les plus intéressés, le site du projet est accessible via **http://tservice.net.ru/~s0mbre/old/?section=projects&item=kevent**. Vous y trouverez les dernières nouveautés concernant Kevent, une documentation en cours d'écriture, ainsi que des *benchmarks*. Aussi, la mise à disposition d'exemples d'utilisation est prévue lorsque l'API sera complètement stabilisée, c'est-à-dire lorsque Kevent sera inclus dans la mainline, ce qui semble être un événement proche ;)

KPAGE, UN MODULE POUR EXPLORER LA MÉMOIRE

Frédéric Raynal – fred@security-labs.org

Kpage **[1]** est un petit module développé à l'occasion pour expliquer le comportement du bit NX sur les architectures 64 bits. Il permet de se promener dans l'organisation des pages mémoires. Pour y parvenir, on utilise plusieurs « objets » et « techniques » liés à la programmation noyau : voici ce que propose d'explorer cet article.

Ce petit module de rien du tout touche en réalité à plusieurs choses amusantes :

- d'abord son objectif est d'afficher la structure des pages jusqu'au descripteur, voire le contenu de ces pages : il s'agit donc de manipuler directement la mémoire, mieux vaut éviter de se planter, ce serait fatal ;
- il faut pouvoir accéder à ces informations depuis l'espace utilisateur : on crée donc une sous-arborescence dans le `/proc` pour cela ;
- on souhaite pouvoir changer le PID et l'adresse à explorer sans avoir à charger/décharger le module à chaque fois.

La compilation facile avec kbuild

La compilation des modules est devenue très simple depuis l'adoption de kbuild.

```
$ cat Makefile
EXTRA_CFLAGS := -D_X86_64_
#EXTRA_CFLAGS := -D_X86_32_
ifneq ($(KERNELRELEASE),)
```

KPAGE, UN MODULE POUR EXPLORER LA MÉMOIRE

Frédéric Raynal – fred@security-labs.org

```
obj-m   := kpage.o
kpage-y := kparam.o pagetable.o registers.o \
           xdump.o rdump.o kpage_core.o
else
KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)
default:
        $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
clean:
        rm -rf *~ *o *.mod.c .*cmd .tmp_versions
endif
```

En gros, il faut juste retenir que `obj-m` indique que nous allons construire un module, et `kpage-y` liste les fichiers nécessaires pour ça. Ensuite, la ligne avec `$(MAKE)` fait tout le travail pour nous : ça appelle le système kbuild, qui initialise tout ce qui est nécessaire, puis ça revient dans le répertoire courant pour construire ce qu'on lui demande :

```
raynal@batman:~/MISC/kpage/kpage$ make
make -C /lib/modules/2.6.18-3-amd64/build
SUBDIRS=/home/raynal/MISC/kpage/kpage modules
make[1]: Entering directory `/usr/src/linux-headers-2.6.18-3-amd64'
  CC [M]  /home/raynal/MISC/kpage/kpage/kparam.o
  CC [M]  /home/raynal/MISC/kpage/kpage/pagetable.o
  CC [M]  /home/raynal/MISC/kpage/kpage/registers.o
  CC [M]  /home/raynal/MISC/kpage/kpage/xdump.o
  CC [M]  /home/raynal/MISC/kpage/kpage/rdump.o
  CC [M]  /home/raynal/MISC/kpage/kpage/kpage_core.o
  LD [M]  /home/raynal/MISC/kpage/kpage/kpage.o
  Building modules, stage 2.
  MODPOST
  CC      /home/raynal/MISC/kpage/kpage/kpage.mod.o
  LD [M]  /home/raynal/MISC/kpage/kpage/kpage.ko
make[1]: Leaving directory `/usr/src/linux-headers-2.6.18-3-amd64'
```

Du côté des sources, plus besoin de déclarations étranges signalant qu'il s'agit de code noyau ou d'un module, tout cela est pris en charge par kbuild.

Se promener dans les pages mémoires

Il ne s'agit pas ici d'un cours sur la mémoire, mais de voir les structures et les méthodes de programmation associées. Donc, nous rappellerons juste que la pagination est un mécanisme qui intervient après la segmentation. La segmentation n'est pas ou peu utilisée, et repose sur le modèle *flat* en général, c'est-à-dire qu'à une adresse logique correspond une adresse linéaire, à une transformation linéaire près. Le modèle flat fait en sorte que cette correspondance soit directe (ou presque). Nous n'en parlerons donc plus.

L'objectif de la pagination est de convertir cette adresse linéaire en une adresse physique, c'est-à-dire retrouver l'endroit où se cachent les octets correspondant à cette adresse. Pour cela, cette adresse est décomposée en plusieurs blocs, chaque bloc étant en réalité un index dans un tableau.

Linux supporte un nombre non négligeable d'architectures. Manque de chance, certaines utilisent une pagination à 4 niveaux (x86_64 comme nous le verrons par la suite), d'autres à 2 niveaux (IA32 en mode normal) et d'autres, pas du tout de pagination (puisqu'elle est optionnelle). Histoire de se simplifier la vie, les développeurs ont donc considéré qu'il y avait toujours 4 niveaux de pagination (enfin, quand pagination il y a). La seule chose qu'ils changent, ce sont les macros permettant de passer d'un niveau au suivant.

Les 4 niveaux sont donc : Page Global Directory (PGD), Page Upper Directory (PUD), Page Middle Directory (PMD) et, enfin, Page Table Entry (PTE). Il faut voir chacune de ces structures comme des tableaux dont les cases pointent sur un répertoire du niveau suivant. Ainsi, une entrée du PGD pointe sur l'élément 0 d'un PUD. Sur x86_64, il y a au plus 512 entrées dans le PGD, mais toutes ne sont pas utilisées en même temps. En gros, celles utilisées vont permettre de découper la mémoire pour les régions allouées (noyau, pile, tas, les bibliothèques, le binaire, etc.). Une entrée du PGD nous conduit à un PUD, composé lui aussi de 512 entrées sur x86_64. Chaque entrée du PUD pointe sur un PMD, et ainsi de suite. Au dernier niveau, chaque entrée PTE contient un descripteur de la page physique, descripteur qui fournit de précieuses informations sur la page physique :

- Le niveau de privilège requis pour y accéder, rien à voir avec les rings de la segmentation, il n'y a là que deux niveaux : `user` ou `superviser`, qui correspondent respectivement au mode utilisateur ou noyau.
- La page est-elle accessible en lecture/écriture ou lecture seule (bit `RW`) ?
- La page a-t-elle été modifiée depuis qu'elle est en mémoire (bit `dirty`) ?
- La page est-elle présente en mémoire (bit `present`) ?
- Etc.

La pagination permet d'éviter la redondance des pages en mémoire. Nous verrons comment/pourquoi par la suite, mais le principe est que si une bibliothèque, par exemple, est utilisée par plusieurs processus, ses instructions seront dans quelques pages physiques, une bonne fois pour toute. Ensuite, les processus qui en ont besoin les utiliseront, sans doute à des adresses virtuelles différentes, mais qui pointeront toutes vers ces mêmes pages physiques.

Du coup, on réalise que tout ce qui s'exécute quand la pagination est activée a besoin de son propre environnement de pagination. Et comme nous l'avons dit, la base, c'est le PGD. Pour cela, la convention veut que le registre CR3 contienne l'adresse du PGD (sur x86) : chaque tâche possède son propre PGD, et à chaque changement de contexte, le registre CR3 est mis à jour. Mais il est une « tâche » à laquelle

KPAGE, UN MODULE POUR EXPLORER LA MÉMOIRE

Frédéric Raynal – fred@security-labs.org

on ne pense pas systématiquement : le noyau. Lui aussi possède son propre PGD, et donc les pages physiques qui vont avec.

Si on prend le découpage classique de l'espace mémoire, les adresses inférieures à 0xc0000000 (3 Go) correspondent à l'utilisateur, et le dernier Giga au noyau. On retrouve ce découpage au niveau du PGD où les premières entrées correspondent à l'espace utilisateur, et les dernières à l'espace noyau. Mais l'espace noyau est commun à tous les processus, donc tous les processus ont des entrées communes pour cette partie de la mémoire. Elles correspondent à ce qu'on appelle le *master kernel PGD* (son adresse est placée dans la variable `swapper_pg_dir`). Derrière ce nom pompeux, se cache simplement le PGD du noyau, c'est-à-dire les pages physiques utilisées par le noyau lui-même. Ainsi, selon le contexte d'exécution, on sait toujours à quel PGD se référer : `swapper_pg_dir` en mode noyau, celui de la tâche en mode utilisateur (même si celui de la tâche, répétons-le, contient des entrées sur les pages noyau).

Selon les architectures, certains de ces niveaux se contentent de passer la main au niveau suivant. Ainsi, sur x86_64, on a l'organisation suivante :

```
// asm-x86_64/pgtable.h
/*
 * PGDIR_SHIFT determines what a top-level page table
 * entry can map
 */
#define PGDIR_SHIFT     39
#define PTRS_PER_PGD    512
/*
 * 3rd level page
 */
#define PUD_SHIFT       30
#define PTRS_PER_PUD    512
/*
 * PMD_SHIFT determines the size of the area a middle-level
 * page table can map
 */
#define PMD_SHIFT       21
#define PTRS_PER_PMD    512
/*
 * entries per page directory level
 */
#define PTRS_PER_PTE    512
#define pgd_index(address)
     (((address) >> PGDIR_SHIFT) & (PTRS_PER_PGD-1))
#define pud_index(address)
     (((address) >> PUD_SHIFT) & (PTRS_PER_PUD-1))
#define pmd_index(address)
     (((address) >> PMD_SHIFT) & (PTRS_PER_PMD-1))
```

En revanche, sur i386 en mode standard, la pagination se fait sur 2 niveaux uniquement (contre 3 pour le mode PAE). Nous ne détaillerons pas plus le fonctionnement et l'organisation des structures liées à la pagination, alors entrons (enfin) dans le vif du sujet.

Petit voyage au centre du noyau ou notre module vu de dehors

Dans la suite, nous donnerons du code simplifié, sans la gestion des erreurs ou quelques autres aspects « secondaires ». Nous nous appuierons sur l'exemple suivant :

```
$ sudo insmode ./kpage.ko
$ cat /proc/`pidof top`/maps
00400000-0040d000 r-xp 00000000 09:02 130982          /usr/bin/top
0050d000-0050e000 rw-p 0000d000 09:02 130982          /usr/bin/top
0050e000-00532000 rw-p 0050e000 00:00 0               [heap]
2b5cdea1e000-2b5cdea35000 r-xp 00000000 09:01 482137  /lib/ld-2.3.6.so
2b5cdea35000-2b5cdea38000 rw-p 2b5cdea35000 00:00 0
2b5cdeb34000-2b5cdeb36000 rw-p 00016000 09:01 482137  /lib/ld-2.3.6.so
2b5cdeb36000-2b5cdeb43000 r-xp 00000000 09:01 482065  /lib/libproc-3.2.7.so
2b5cdeb43000-2b5cdec43000 ---p 0000d000 09:01 482065  /lib/libproc-3.2.7.so
2b5cdec43000-2b5cdec44000 rw-p 0000d000 09:01 482065  /lib/libproc-3.2.7.so
...
2b5cdf529000-2b5cdf52b000 rw-p 00009000 09:01 481957  /lib/libnss_files-2.3.6.so
7fffcc076000-7fffcc08c000 rw-p 7fffcc076000 00:00 0   [stack]
ffffffffff600000-ffffffffffe00000 ---p 00000000 00:00 0 [vdso]
```

Le chargement du module crée une nouvelle entrée dans `/proc` :

```
$ ls /proc/kpage
cr4  efer  param  pgd  pmd  pte  pud  rdump  xdump
```

`cr4` et `efer` affichent le contenu de registres qui ne sont normalement accessibles qu'en mode privilégié. `param` est l'entrée que nous utilisons pour passer les paramètres à notre module (PID et adresse). `pgd`, `pud`, `pmd` et `pte` affichent les tableaux complets, ainsi que les informations relatives à l'adresse passée en argument. Enfin, `rdump` et `xdump` donnent accès aux pages mémoires directement.

On examine dans le `/proc` l'organisation de sa mémoire pour déterminer quelle adresse nous préoccupe. On pourrait prendre n'importe laquelle dans l'espace d'adressage, mais autant en prendre une valide. Notre choix porte sur `0x7fffcc08a000` qui ne se trouve pas trop loin du sommet de la pile, et devrait donc contenir des informations :

```
$ sudo echo `pidof top` 0x7fffcc08a000 > /proc/kpage/param
```

Nous pouvons maintenant aller lire les différentes entrées correspondant à l'adresse choisie :

```
$ sudo cat /proc/kpage/pgd /proc/kpage/pud /proc/
kpage/pmd /proc/kpage/pte
BASE PGD=0xffff81000ffc2000 => idx=86  current=0xfff
f81000ffc22b0 (pgd_k=0xffff81002ea28000)
   ffff81000ffc2000   0 000: Present=1 R/W=W U/S=U
Access=1 PhyPgSz=4k(pte) base=0x0000000065259000 NX=0
-> ffff81000ffc22b0  86 056: Present=1 R/W=W U/S=U
Access=1 PhyPgSz=4k(pte) base=0x000000001138f000 NX=0
   ffff81000ffc27f8 255 0ff: Present=1 R/W=W U/S=U
Access=1 PhyPgSz=4k(pte) base=0x000000005cd73000 NX=0
   ffff81000ffc2810 258 102: Present=1 R/W=W U/S=S
Access=1 PhyPgSz=4k(pte) base=0x0000000000008000 NX=0
   ffff81000ffc2c20 388 184: Present=1 R/W=W U/S=U
```

KPAGE, UN MODULE POUR EXPLORER LA MÉMOIRE

Frédéric Raynal – fred@security-labs.org

```
Access=1 PhyPgSz=4k(pte) base=0x00000000bd5d8000 NX=0
   ffff81000ffc2ff8 511 1ff: Present=1 R/W=W U/S=U
Access=1 PhyPgSz=4k(pte) base=0x0000000000203000 NX=0
BASE PUD=0xffff81001138f000 => idx=371 current=0xfff
f81001138fb98
-> ffff81001138fb98 371 173: Present=1 R/W=W U/S=U
Access=1 PhyPgSz=4k(pte) base=0x0000000066445000 NX=0
BASE PMD=0xffff810066445000 => idx=250 current=0xfff
f8100664457d0
   ffff8100664457a8 245 0f5: Present=1 R/W=W U/S=U
Access=1 PhyPgSz=4k(pte) base=0x000000003b000000 NX=0
   ffff8100664457b0 246 0f6: Present=1 R/W=W U/S=U
Access=1 PhyPgSz=4k(pte) base=0x00000001aa03000 NX=0
   ffff8100664457b8 247 0f7: Present=1 R/W=W U/S=U
Access=1 PhyPgSz=4k(pte) base=0x0000000092d18000 NX=0
   ffff8100664457c0 248 0f8: Present=1 R/W=W U/S=U
Access=1 PhyPgSz=4k(pte) base=0x0000000087420000 NX=0
   ffff8100664457c8 249 0f9: Present=1 R/W=W U/S=U
Access=1 PhyPgSz=4k(pte) base=0x0000000039ba7000 NX=0
-> ffff8100664457d0 250 0fa: Present=1 R/W=W U/S=U
Access=1 PhyPgSz=4k(pte) base=0x000000071a0f000 NX=0
BASE PTE=0xffff810071a0f000 => idx=297 (129) current
=0xffff810071a0f948
   ffff810071a0f0f0  30 01e: Present=1 R/W=W U/S=U
Access=1 Dirty=1 Glob=0 base=0x0000000077892000 NX=1
   ffff810071a0f0f8  31 01f: Present=1 R/W=W U/S=U
Access=1 Dirty=1 Glob=0 base=0x00000000562bc000 NX=1
   ffff810071a0f100  32 020: Present=1 R/W=R U/S=U
Access=1 Dirty=0 Glob=0 base=0x00000000bb84e000 NX=0
   ffff810071a0f108  33 021: Present=1 R/W=R U/S=U
Access=1 Dirty=0 Glob=0 base=0x00000000bb84f000 NX=0
   ffff810071a0f110  34 022: Present=1 R/W=R U/S=U
Access=1 Dirty=0 Glob=0 base=0x00000000bca6c000 NX=0
   ffff810071a0f140  40 028: Present=1 R/W=R U/S=U
Access=1 Dirty=0 Glob=0 base=0x00000000bb818000 NX=0
-> ffff810071a0f948 297 129: Present=1 R/W=W U/S=U
Access=1 Dirty=1 Glob=0 base=0x000000005a1d6000 NX=1
   ffff810071a0f950 298 12a: Present=1 R/W=W U/S=U
Access=1 Dirty=1 Glob=0 base=0x0000000066fe6000 NX=1
```

Le filesystem /proc

Toute l'interaction avec notre module se fait dans le /proc. Il s'agit d'un système de fichiers virtuel. Pour cela, on crée le répertoire (fonction proc_mkdir()), puis on le peuple avec nos fichiers (fonction create_proc_entry()) :

```
//On crée le répertoire
kp_dir = proc_mkdir(MODULE_NAME, NULL);
kp_dir->owner = THIS_MODULE;

// On crée l'entrée param pour passer les arguments
kp_param = create_proc_entry( "param", DEFAULT_RW_
PERMS, kp_dir);
kp_param->owner = THIS_MODULE;
kp_param->read_proc = kpage_read_param;
kp_param->write_proc = kpage_write_param;

// Idem pour le PGD
kp_pgd = create_proc_entry( "pgd", DEFAULT_R_PERMS,
kp_dir);
kp_pgd->owner = THIS_MODULE;
kp_pgd->proc_fops = &pgd_operations;

// Et ainsi de suite ...
```

Pour chaque entrée, on définit les opérations associées. Dans le cas de param, on utilise directement les pointeurs sur les fonctions {read|write}_proc. En revanche, pour le PGD, on passe par une structure struct file_operations *proc_fops plus générale qui permet d'un coup de définir toutes les opérations (voir include/linux/proc_fs.h). Regardons maintenant ces différences.

La gestion des paramètres

L'entrée param nous sert à passer les arguments à notre module. Il nous faut donc lire et écrire dans cette entrée, et définir les opérations associées.

On utilise un simple echo pour passer nos arguments dans /proc/param. Derrière cela, se cache la fonction kpage_write_param() qui lit ce qui est écrit, et l'utilise pour initialiser les données internes :

```
int kpage_write_param( struct file *filp, const char __user *buff,
                       unsigned long len, void *data )
{
    char cmdline[CMDLINE_LENGTH];
    struct task_struct *p;

    ...
    /* passage du user space dans le kernel space */
    if (copy_from_user( &cmdline, buff, len )) {
        return -EFAULT;
    }
    cmdline[len] = '\0';

    ...
    /* Plein d'opérations sur les structures internes du module pour
       sauvegarder les éléments */
    memset(&kparam, 0, sizeof(kparam));
    kparam.dlen = PAGE_SIZE;
    if ( (sscanf(cmdline, "%lu %lx %ld",
     (unsigned long*)&kparam.pid, &kparam.vaddr, &kparam.dlen) == 0)
     || (kparam.pid == 0) || (kparam.vaddr == 0) ) {
        printk(KERN_ERR "kpage: invalid parameter(s) in cmdline.\n");
        return -EINVAL;
    }
    kparam.pgd = pgd_offset(p->mm, kparam.vaddr);
    if (pgd_none(*kparam.pgd))
        return len;

    ...
    return len;
}
```

Quand on lit l'entrée param, on affiche les informations sur l'adresse précédemment écrite dans param (ici, on a changé par rapport à l'adresse précédente) :

```
$ sudo cat /proc/kpage/param
PID = 6378
vaddr = 0x00002b5cdf529000
pgd idx = 056 00002b0000000000
pud idx = 173 0000005cc0000000
pmd idx = 0fa 00000000lf400000
pte idx = 129 0000000000129000
dlen = 4096
```

KPAGE, UN MODULE POUR EXPLORER LA MÉMOIRE

Frédéric Raynal – fred@security-labs.org

Pour cela, la fonction de lecture, `kpage_read_param()`, affiche simplement les informations en allant les chercher où elles sont. Cette fonction doit retourner le nombre de caractères écrits :

```
int kpage_read_param( char *page, char **start, off_t off,
                      int count, int *eof, void *data )
{
    len += sprintf(page+len, "PID = %lu\n"
                   "vaddr = 0x%016lx\n"
                   "pgd idx = %03lx %016lx\n"
                   "pud idx = %03lx %016lx\n"
                        ...
                   (unsigned long)kparam.pid,
                   kparam.vaddr,
                   pgd_index(kparam.vaddr),
                        ...
                        );
  return len;
}
```

Le problème de cette approche est que le buffer utilisé pour écrire/lire est de taille finie, voire insuffisante. Si on lit/écrit plus de 1024 caractères, le buffer étant circulaire, on revient au début (enfin, tout ceci n'est pas strictement vrai, mais c'est le principe). Nous ne pouvons donc pas utiliser ces opérations directes de lecture/écriture pour les entrées sur les pages, car on risque de dépasser largement les 1024 caractères. Par exemple, si on prend les entrées `rdump` et `xdump`, elles affichent des pages complètes, soit 4 Ko. Pour cela, on utilise une autre méthode, itérative, fournie par le noyau.

Lecture itérative

Nous nous concentrons sur le cas du PGD, les autres étant similaires. Cette fois, nous devons donc définir une fonction itérative associée à la lecture de l'entrée :

```
int pgd_show(struct seq_file *p, void *v)
{
    int i =  *(loff_t *) v;
    ...
    seq_printf(p, "%s %p ",
      i == pgd_index(kparam.vaddr) ? "->" : "  ",
      &t->mm->pgd[i]);
    ...
    return 0;
}
```

Ce bref code se concentre sur l'affichage. Tout d'abord, les arguments de la fonction sont tout d'abord un fichier dans lequel on écrit, puis un pointeur sur `void`. Le fichier n'est pas un fichier classique, mais un fichier séquentiel qui permet des opérations « par bloc » (`include/linux/seq_file.h`). En fait, il s'agit d'un fichier muni d'un itérateur :

```
// include/linux/seq_file.h
struct seq_operations {
  void * (*start) (struct seq_file *m, loff_t *pos);
  void (*stop) (struct seq_file *m, void *v);
  void * (*next) (struct seq_file *m, void *v, loff_t *pos);
  int (*show) (struct seq_file *m, void *v);
};
```

En définissant ces opérations, on bénéficie d'une lecture séquentielle du fichier. Or, jusqu'à présent, nous avons uniquement défini la fonction `show()`. Les autres sont celles qui gèrent l'itérateur :

- `start()` sert à renvoyer l'index à traiter ou NULL si on est au-delà de ce qu'on souhaite traiter ;
- `next()` gère l'incrémentation du compteur ;
- `stop()`, on s'en moque, car on gère ça avant.

Comme on va faire exactement les mêmes choses pour les PGD, PUD, PMD et PTE, et qu'en plus on est flemmard, on va utiliser des macros pour tout faire d'un coup :

```
#define PGTABLE_SEQ_ITER(pgtable, limit) \
    static void * pgtable##_seq_start(struct seq_file *f, loff_t *pos) \
    {                                                                  \
        return (*pos < limit ) ? pos : NULL;                           \
    }                                                                  \
    static void * pgtable##_seq_next(struct seq_file *f, void *v, loff_t *pos) \
    {                                                                  \
        (*pos)++;                                                      \
        if (*pos >= limit)                                             \
                return NULL;                                           \
        return pos;                                                    \
    }                                                                  \
    static void pgtable##_seq_stop(struct seq_file *f, void *v)        \
    {                                                                  \
        /* Nothing to do */                                            \
    }                                                                  \
                                                                       \
    static struct seq_operations pgtable##_seq_ops = {                 \
        .start = pgtable##_seq_start,                                  \
        .next  = pgtable##_seq_next,                                   \
        .stop  = pgtable##_seq_stop,                                   \
        .show  = pgtable##_show                                        \
    };                                                                 \
    static int pgtable##_open(struct inode *inode, struct file *filp)  \
    {                                                                  \
        return seq_open(filp, & pgtable##_seq_ops);                    \
    }                                                                  \
      struct file_operations pgtable##_operations = {                  \
        .open           = pgtable##_open,                              \
        .read           = seq_read,                                    \
        .llseek         = seq_lseek,                                   \
        .release        = seq_release,                                 \
    };
PGTABLE_SEQ_ITER(pgd, PTRS_PER_PGD)
PGTABLE_SEQ_ITER(pud, PTRS_PER_PUD)
PGTABLE_SEQ_ITER(pmd, PTRS_PER_PMD)
PGTABLE_SEQ_ITER(pte, PTRS_PER_PTE)
```

Grâce au `##`, qui sert à la concaténation, on définit très vite toutes nos fonctions, et on remplit les structures pour chaque type de tableau de pages. Et voilà, on sait maintenant lire/écrire des entrées, soit par la méthode classique, soit en mode itératif.

Petit voyage au centre du noyau ou notre module vu de dedans

Puisque nous voulons parcourir les différents répertoires de pages, commençons par le commencement, le PGD. En fait, chaque tâche possède son propre PGD, c'est ce qui permet de distinguer quelles pages

KPAGE, UN MODULE POUR EXPLORER LA MÉMOIRE

Frédéric Raynal – fred@security-labs.org

appartiennent à qui. À chaque changement de tâche, le registre CR3 est mis à jour, avec l'adresse du PGD correspondant à la tâche qui va s'exécuter.

Or, comme nous sommes dans le noyau à exécuter le code de notre module, CR3 contient l'adresse du PGD... du noyau, et non de la tâche souhaitée. En fait, ce PGD est sauvegardé dans le descripteur des tâches, `task_struct` (`include/linux/sched.h`). Ce descripteur contient un pointeur sur une structure qui décrit complètement la mémoire de la tâche (`mm_struct`, toujours dans `sched.h`), qui à son tour contient l'adresse du PGD de la tâche.

Comment y accéder : il faut d'abord récupérer le descripteur de tâche, puis accéder aux différents éléments au travers des structures impliquées :

```
int pgd_show(struct seq_file *p, void *v)
{
    int i =  *(loff_t *) v;
    struct task_struct *t = find_task_by_pid(kparam.pid);

    __asm__ __volatile__("mov %%cr3, %0"
                         : "=r"(cr3)
        );
    /* Affiche les info relatives a notre
       adresse sur la premiere ligne */
    if (0 == i) {
      seq_printf(p, "BASE PGD=0x%p => idx=%-3ld "
                    "current=0x%p (pgd_k=0x%lx) \n",
       t->mm->pgd, pgd_index(kparam.vaddr),
       kparam.pgd, cr3+PAGE_OFFSET
      );
    }
   /* Affiche chaque ligne de la PGD */
    if ( !pgd_none(t->mm->pgd[i]) ) {
        seq_printf(p, "%s %p ",
               i == pgd_index(kparam.vaddr) ? "->" : "  ",
               &t->mm->pgd[i]);
        dump_pte(p, i, pgd_val(t->mm->pgd[i]), 0);
    }
    return 0;
}
```

Quelques remarques sur le code qui précède :

- On met de l'assembleur *inline* pour récupérer le registre CR3 qui contient le PGD du noyau.
- On utilise la lecture itérative pour faire un affichage spécial si `0==i`. Remarquez l'écriture, avec le `0` à gauche : c'est une habitude à prendre qui permet d'éviter les malencontreux oublis d'un signe `=`. En effet, si vous écrivez `i=0` dans une condition, le compilateur ne se plaindra pas et votre programme risque d'avoir un comportement non désiré. En revanche, `0=i` provoque une erreur du compilateur.
- Enfin, s'il y a une entrée dans le PGD (`pgd_none()`), on affiche les informations relatives en appelant la fonction générique `dump_pte()`.
- La macro `pgd_index()` convertit les bits de l'adresse virtuelle en l'index dans la PGD (on trouve les mêmes fonctions pour les autres niveaux).

Revenons quelques instants sur la pagination. Pour se promener dans les répertoires, on doit à chaque niveau, accéder à l'entrée du répertoire courant, qui pointe sur la table suivante. Regardons comment cela fonctionne au niveau du PUD :

```
// kparam.vaddr contient l'adresse virtuelle
// on récupère l'index à partir de l'adresse
idx = pud_index(kparam.vaddr);

// On récupère la base du PUD à partir du PGD et de l'adresse
kparam.pud = pud_offset(kparam.pgd, kparam.vaddr);

// on remonte du PUD courant à la base
pud = kparam.pud - idx;
```

En fait, `pud_offset()` nous donne directement le bon PUD. Pour passer du PGD à la base du PUD suivant, on devrait faire (écriture symbolique) :

```
pgd = cr3
pud = pgd[ pgd_index(vaddr) ]
pmd = pud[ pud_index(vaddr) ]
pte = pmd[ pmd_index(vaddr) ]
```

Et quand on met tout ça ensemble : rdump et xdump

Ces 2 entrées sont destinées à *dumper* les pages mémoires des processus. `xdump` affiche conjointement le code hexa et sa transcription en ascii de chaque octet. `rdump` fait un affichage raw, ce qui est très pratique pour combiner avec d'autres programmes (par exemple, pour désassembler un programme protéger, en « pipant »dans `ndisasm`).

```
$ echo `pidof top` 0x7fffcc08a000 > /proc/kpage/param
$ cat /proc/kpage/xdump
Dumping vaddr=0x00007fffcc08a000
paddr=0x000000000baf6b000
...
00007fffcc08ae90: 00 50 41 54 48 3d 2f 75  73 72 2f
6c 6f 63 61 6c  .PATH=/usr/local
00007fffcc08aea0: 2f 73 62 69 6e 3a 2f 75  73 72 2f
6c 6f 63 61 6c  /sbin:/usr/local
00007fffcc08aeb0: 2f 62 69 6e 3a 2f 75 73  72 2f 73
62 69 6e 3a 2f  /bin:/usr/sbin:/
00007fffcc08aec0: 75 73 72 2f 62 69 6e 3a  2f 73 62
69 6e 3a 2f 62  usr/bin:/sbin:/b
00007fffcc08aed0: 69 6e 3a 2f 75 73 72 2f  58 31 31
52 36 2f 62 69  in:/usr/X11R6/bi
00007fffcc08aee0: 6e 00 53 54 59 3d 36 33  36 30 2e
73 68 00 50 57  n.STY=6360.sh.PW
00007fffcc08aef0: 44 3d 2f 68 6f 6d 65 2f  72 61 79
6e 61 6c 2f 4d  D=/home/raynal/M
00007fffcc08af00: 49 53 43 2f 6b 70 61 67  65 2f 6b
70 61 67 65 00  ISC/kpage/kpage.
00007fffcc08af10: 4c 41 4e 47 3d 65 6e 5f  55 53 2e
49 53 4f 2d 38  LANG=en_US.ISO-8
00007fffcc08af20: 38 35 39 2d 31 35 00 48  4f 4d 45
3d 2f 68 6f 6d  859-15.HOME=/hom
00007fffcc08af30: 65 2f 72 61 79 6e 61 6c  00 53 55
44 4f 5f 43 4f  e/raynal.SUDO_CO
00007fffcc08af40: 4d 4d 41 4e 44 3d 2f 62  69 6e 2f
62 61 73 68 00  MMAND=/bin/bash.
00007fffcc08af50: 53 48 4c 56 4c 3d 32 00  4c 41 4e
47 55 41 47 45  SHLVL=2.LANGUAGE
...
```

KPAGE, UN MODULE POUR EXPLORER LA MÉMOIRE

Frédéric Raynal – fred@security-labs.org

On passe par une lecture itérative étant donnée la quantité d'octets à retranscrire. On commence par donner l'adresse physique correspondant à l'adresse virtuelle passée en argument. Ensuite, on navigue au sein des répertoires de page pour atteindre la vraie page physique, et afficher son contenu :

```
char *ptr;
pte = lookup_vaddr(kparam.pgd, vaddr);
ptr = (char*)( (pte_val(*pte)&PTE_MASK)) +
   PAGE_OFFSET + (kparam.vaddr&~PAGE_MASK);
```

Comment accéder réellement aux octets de la page physique ? En fait, on peut y accéder au travers de la mémoire du noyau qui contient un *mapping* direct pour cela. On récupère donc l'adresse physique à partir du PTE, via l'opération `pte_val(*kparam.pte)&PTE_MASK`. Mais on ne peut pas utiliser directement cette adresse physique pour atteindre la page puisque le noyau ne sait utiliser que des adresses virtuelles. On doit donc « reconvertir » cette adresse en une adresse virtuelle. Mais pas n'importe laquelle ! On va utiliser une fonctionnalité du noyau, qui contient son propre mapping de TOUTES les pages physiques. En effet, pour accéder à une adresse physique depuis le noyau, il suffit d'y ajouter `PAGE_OFFSET`. Ensuite, on extrait l'offset de l'adresse qu'on veut, comme on le fait pour les index des tables de répertoire (l'offset correspond aux derniers bits de l'adresse virtuelle). Ainsi, `ptr` est une adresse virtuelle qui pointe sur la bonne page physique.

Un des intérêts de ce mécanisme est que si plusieurs processus sont des instances du même programme (un *shell* par exemple), certaines pages ne sont chargées qu'une fois en mémoire. Les adresses virtuelles seront très probablement différentes. Mais les pages physiques correspondantes seront les mêmes tant qu'elles n'auront pas été modifiées.

Last words...

Le mode noyau donne une compréhension très précise de ce qui se passe. Programmer en mode noyau n'est pas simple, d'autant qu'il y a énormément de contraintes auxquelles nous ne sommes pas habitués : passage des arguments entre espace utilisateur et espace noyau, gestion des conditions de concurrence, allocateurs mémoires, *dead locks*, etc. Il s'agit réellement d'un autre monde. C'est ce qui fait que c'est sans doute à la fois si compliqué de s'y aventurer, mais aussi tellement intéressant !!!

Nous avons vu comment un petit module a priori tout simple pouvait mettre en jeu de nombreux mécanismes qu'on utilise parfois sans s'en rendre compte (le `/proc`, la conversion d'adresse via la pagination). Faire ses propres programmes de test n'est pas si compliqué. On risque juste de cramer sa machine ;-) Utilisez toujours une machine virtuelle pour faire vos tests, c'est beaucoup plus sûr. En effet, si vous vous ratez en mode noyau, ce n'est pas un processus qui plante, c'est le noyau, avec des effets imprévisibles potentiellement destructeurs...

[1] kpage, à télécharger sur **http://miscmag.com/files/28/kpage/**

LINUX : STABILITÉ ET PILOTES, PEUT-ON AMÉLIORER LES CHOSES ?

Matthieu Barthélemy – bonsouere@gmail.com

Notre noyau favori est remarquable en plusieurs points ; apprécié pour sa robustesse et sa fiabilité, il tourne sur une variété de plateformes impressionnante, gère une base de matériel record et est développé par un très grand nombre de contributeurs. Ce qui fait sa force représente aussi une de ses faiblesses : le code afférant à la gestion des périphériques représente environ 70% du volume de code total de Linux, et, selon une étude de l'université de Stanford, Californie, il serait à l'origine des 3 à 7 fois plus de plantages entraînant un crash de l'ensemble du noyau que les autres portions du code (mesures automatiques effectuées sur des noyaux 2.0 à 2.4). D'autres mesures effectuées cette fois sur Windows XP concluent que 85% des plantages sont dus aux pilotes.

Une question d'architecture

Mais pourquoi Linux est-il si sensible à l'influence de pilotes ? Qu'en est-il ailleurs ? C'est principalement une question de design. On peut ainsi distinguer plusieurs catégories de noyaux existants, dont voici les principales :

- Les micro-noyaux : dans ce cas, un cœur minimaliste assure les fonctions de base (gestion de la mémoire, des tâches, accès basique au matériel...). Ce sont d'autres modules qui assurent le reste (pilotes, systèmes de fichiers, réseau...), en espace utilisateur. Mach ou encore L4 (HURD) en sont des exemples. HURD en particulier est conçu pour que l'on puisse altérer, ajouter et retirer ces modules sans affecter du tout le reste du système ; ils communiquent entre eux au moyen d'interfaces RPC (*Remote Procedure Call*), et sont chacun isolés dans leur zone mémoire, sans pouvoir en déborder. Le principal inconvénient de cette approche est le coût des communications entre les différentes parties du système, et entre les espaces noyau et utilisateur. En effet, si

LINUX : STABILITÉ ET PILOTES, PEUT-ON AMÉLIORER LES CHOSES ?

Matthieu Barthélemy – bonsouere@gmail.com

chaque composant du système forme une entité indépendante, chaque appel à l'un d'entre eux, chaque communication entre modules requiert des *context switches* (changement de contexte) : sauvegarde des registres processeur concernant la tâche en cours, chargement dans les registres des informations de la tâche suivante, et ainsi de suite ; ce type d'opération est coûteux en cycles CPU, spécialement sur architectures x86.

- Les noyaux monolithiques : le noyau est un ensemble regroupant toutes les fonctions telles que gestion des processus, gestion mémoire, pilotes de périphériques, systèmes de fichiers... Les différentes parties du système ne sont pas isolables et sont fortement couplées les unes aux autres. Linux et les BSD sont des noyaux monolithiques. Ces systèmes offrent de bonnes performances, mais chaque partie peut en théorie affecter le reste du système, étant donné qu'il n'y a pas de protections (mémoire, routines et structures internes).
- Les noyaux hybrides : ce sont à la base des micro-noyaux, mais pour offrir un bon compromis de performances, le cœur du noyau se charge de tâches normalement dévolues aux services en espace utilisateur dans les micro-noyaux purs (accès au matériel...). C'est le choix retenu par les noyaux NT (Microsoft) et XNU (OSX).

Linux, ainsi que d'autres systèmes, ne sont en fait plus strictement monolithiques, mais monolithiques modulaires. Ainsi, sous notre OS, il est possible depuis longtemps de rajouter et retirer des objets noyau (tout code de type *kernel object*), que nous appelons couramment modules (pensez aux célèbres `insmod/modprobe/rmmod`), ou encore de modifier les gestionnaires d'entrées/sorties et de tâches en cours d'exécution. Cependant, ces fonctionnalités ont été intégrées dans un but de gain de place et de souplesse, en ne chargeant que du code utile à un système donné. Cette organisation est défendue par Linus Torvalds, qui s'est opposé depuis longtemps au principe de micro-noyau. Cependant les réflexions vont bon train depuis de nombreuses années pour trouver des solutions alliant la performance d'une approche monolithique et la robustesse d'une approche entièrement modulaire. En fait, dans la partie qui nous intéresse, à savoir améliorer l'indépendance entre le cœur et les pilotes, plusieurs pistes sont explorées par différents projets, chacune ayant ses avantages, mais aucune n'étant parfaite, ni sur le point d'être adoptée.

Pilotes en espace utilisateur

Plusieurs projets ont tenté d'offrir la possibilité de déplacer les pilotes, ou du moins une partie d'entre eux, en espace utilisateur. Il existe déjà de tels pilotes de périphériques (pilotes Xorg, certains drivers USB ou parallèle...), mais qui n'ont pas accès à des fonctions de bas niveau, assurées par le noyau ; c'est pourquoi on ne trouve pas de pilotes en espace utilisateur pour les périphériques connectés sur bus de type PCI. Ainsi, s'il est possible d'accéder aux pages mémoires des périphériques via `/dev/(k)mem`, et de connaître les IRQ et ports d'entrées/sorties qu'ils utilisent, il n'y a aucun moyen de s'enregistrer comme « destinataire » d'une interruption, ni d'être notifié de leur arrivée. Principalement, on peut dire que les derniers remparts à un support du matériel sans intervention du noyau sont la gestion des interruptions (IRQ) et l'accès aux zones DMA. Plusieurs tentatives de solutions ont été proposées pour inclusion dans le noyau. Parmi elles, celle de Peter Chubb en 2004, et, à la dernière rentrée, celle de Thomas Gleixner soutenue par Greg Kroah-Hartman (développeur noyau Suse). Il propose l'ajout d'une nouvelle structure, `iio_device()`, chargée d'enregistrer un nouveau périphérique et de fournir des informations sur ce dernier, ainsi que des fonctions exportées en espace utilisateur (`iio_read`, `iio_write()`...). À charge des développeurs du pilote de créer un module noyau léger, chargé de la gestion des interruptions et de la faire communiquer avec l'*userspace*. Ce choix part du principe que ce travail peut être très spécifique à un périphérique.

La question des pilotes en userspace a déclenché une discussion acharnée sur la LKML. D'un côté, les pilotes développés séparément du noyau seraient plus faciles à écrire, et moins enclins à déstabiliser l'ensemble du système en cas de plantage : étant considérés comme des programmes classiques, ils ne peuvent atteindre les structures du noyau ni son espace mémoire, et peuvent être lancés ou détruits aisément. De l'autre côté, proposer une interface d'accès stable communiquant avec un programme en espace utilisateur ne manquerait certainement pas d'intéresser les vendeurs de matériels, qui pourraient y voir un moyen de développer des pilotes sous Linux sans être soumis à la « contrainte » de les placer sous GPL (rappelons que tout travail directement interfacé au noyau doit se soumettre à la GPL, tandis que l'espace utilisateur n'est pas concerné), ni de suivre les fréquents changements internes des interfaces noyau, ou encore de développer leur propre module intermédiaire en GPL s'interfaçant avec leur BLOB. Par exemple, Linus s'est fermement prononcé contre ce type de projets, à moins qu'on lui démontre que leur utilité dépasse les inconvénients ;

LINUX : STABILITÉ ET PILOTES, PEUT-ON AMÉLIORER LES CHOSES ?

Matthieu Barthélemy – bonsouere@gmail.com

il ne souhaite pas ouvrir une porte supplémentaire aux pilotes binaires propriétaires, ni transformer peu à peu Linux en système à micro-noyau.

Signalons, à titre anecdotique, une approche différente, mais touchant également au domaine des privilèges d'exécution : David Kaplan, étudiant à l'université de l'Illinois (USA), a imaginé et implémenté RingCycle, une solution où les pilotes s'exécutent sur les niveaux de privilèges intermédiaires des processeurs de type x86. Cette architecture définit 4 niveaux de privilèges (ou anneaux, *rings*), un code s'exécutant sur le niveau `n` n'ayant le droit d'intervenir que dans les niveaux égaux ou inférieurs. Linux n'utilise que deux anneaux : 0 pour le noyau et 3 pour l'espace utilisateur. RingCycle propose donc de faire tourner les pilotes dans des *threads* en ring 1, et de faire passer les appels noyau via une API intermédiaire (appelée « Driver API ») au moyen d'appels x86 `lcall` et `lret`. Cette solution nécessite très peu de modifications au code actuel des drivers, mais est réservée à une seule architecture processeur.

La page du projet : **http://www.acm.uiuc.edu/projects/RingCycle**.

SafeDrive et Shadow drivers

Une deuxième piste consiste à tenter d'isoler certaines opérations des pilotes par une couche légère chargée d'intercepter, de vérifier les appels système, voire de décharger-relancer un pilote ayant crashé. Les deux technologies présentées ici ont pour but la fiabilité de fonctionnement en cas d'erreur accidentelle, et ne concernent pas la sécurité ou les codes malveillants.

Les membres du projet SafeDrive de l'université de Berkeley sont partis du constat que ce sont très souvent les requêtes demandées par le système au pilote ou une réception de données inattendue par le pilote qui génèrent des plantages. Le projet a pour but de vérifier la pertinence des données qui sont échangées entre le noyau et un pilote ou encore directement dans un pilote. SafeDrive permet, par exemple, de savoir si l'assignement de variable se fait correctement dans son type et dans sa capacité, par vérification de type des pointeurs et des débordements de tableaux. En pratique, ceci est assuré par la première partie du projet : avant compilation d'un pilote, un outil appelé « Deputy » utilise les informations insérées par les développeurs dans les fichiers d'en-tête (*headers*) ou dans le code pour ajouter au fichier source des lignes de code supplémentaires telles que des appels à `assert` (macro qui renvoie faux si la condition passée en argument n'est pas remplie). On obtient ainsi une vérification à l'exécution, réservée d'ordinaire plutôt aux langages de haut niveau. Par exemple, le code suivant, tiré de la documentation officielle de Safedrive (en rouge, l'annotation du développeur à l'attention de Deputy) :

```
struct e1000_tx_ring {
  [...]
  unsigned int info_count;
  struct e1000_buffer * count(info_count)
      buffer_info;
  [...]
};

static boolean_t
e1000_clean_tx_irq(
    struct e1000_adapter *adapter,
    struct e1000_tx_ring *tx_ring)
{
  [...]
  i = tx_ring->next_to_clean;
  eop = tx_ring->buffer_info[i]
                  .next_to_watch;
}
```

deviendrait, après moulinage par Deputy :

```
struct e1000_tx_ring {
  [...]
  unsigned int info_count;
  struct e1000_buffer *buffer_info;
  [...]
};

static boolean_t
e1000_clean_tx_irq(
    struct e1000_adapter *adapter,
    struct e1000_tx_ring *tx_ring)
{
  [...]
  i = tx_ring->next_to_clean;
  <b>assert(0 <= i && i < tx_ring->info_count);</b>
  eop = tx_ring->buffer_info[i]
                  .next_to_watch;
}
```

La seconde partie est une couche intermédiaire légère entre le noyau et les pilotes. Elle est chargée de mémoriser les dernières actions faites sur le noyau devant être annulées si un pilote plante, et reçoit les appels émis par le code Deputy en cas de problème. Le cas échéant, le module fautif est déchargé, l'état du noyau restauré, et le module peut être rechargé automatiquement afin que le système retrouve un état de fonctionnement complet. Le bon fonctionnement de SafeDrive repose donc sur une préparation du code par ses développeurs. Pour approfondir le sujet, je vous invite à consulter **http://ivy.cs.berkeley.edu/safedrive/**.

Valerie Hansen, développeuse chez Intel, a récemment relancé sur **lwn.net** le concept de *shadow drivers* développé, quant à lui, par l'université de Washington. Cette technique se décompose, à l'instar de SafeDrive, en deux parties. Côté pilotes, chaque catégorie d'entre eux (IDE, réseau...) dispose d'un pseudo-pilote qui est « branché » en parallèle à un « vrai » pilote

LINUX : STABILITÉ ET PILOTES, PEUT-ON AMÉLIORER LES CHOSES ?

Matthieu Barthélemy – bonsouere@gmail.com

et est appelé shadow driver (pilote fantôme). Tant que tout se passe correctement, il est en mode dit « passif » et espionne les appels système (`ioctl()`) en les mémorisant. Si le pilote auquel il est couplé plante, il passe en mode dit « actif », et joue un rôle de proxy : il accepte et répond aux requêtes du noyau comme si de rien n'était. Pendant ce temps-là, le pilote fautif est déchargé, puis rechargé. Les structures noyau allouées par les pilotes sont maintenues, ce qui évite au noyau de remarquer que le pilote n'est plus présent. Les appels faits précédemment au noyau, qui étaient logués en mode passif, sont annulés, afin que celui revienne dans un état cohérent ; notre shadow driver rejoue alors les appels enregistrés au pilote fraîchement rechargé, afin de le remettre dans le même état qu'avant le plantage. Ce dernier recommence alors à traiter les requêtes et à y répondre. Fin de l'incident : le shadow driver retourne en mode passif. Pendant ce temps, aucun message d'erreur n'a donc été transmis aux applications, qui continuent à fonctionner comme si de rien n'était. Ces pilotes fantômes s'appuient sur l'infrastructure Nooks. Nooks est une couche isolant les appels système depuis ou vers les pilotes, et qui détecte les erreurs comme les fautes de protection mémoire, le passage de mauvais paramètres, ou encore un usage CPU trop intensif. Concrètement, chaque driver s'exécute dans un sous-système appelé « *nook* », qui se charge d'intercepter les appels entre noyau et pilote, émule les appels aux registres du périphérique et transfère les interruptions. Chaque nook tourne en espace noyau, mais est un domaine protégé interdisant les débordements mémoires.

L'approche de Nooks + shadow drivers requiert des modifications de code légères pour porter les pilotes existants sur cette infrastructure. À noter que Nooks a été développé sur un noyau 2.4.18, et ne fonctionne en l'état que sur des systèmes monoprocesseurs. La page officielle du projet est http://nooks.cs.washington.edu/

Quelle que soit la solution retenue, l'adoption d'une technique permettant de protéger le cœur du noyau des pilotes aura un coût en ressources système, car elle impliquera, quelle qu'en soit l'implémentation, des vérifications supplémentaires et/ou un code intermédiaire. Cependant, les recherches résumées ici se veulent rassurantes en annonçant un surcoût minime. Pour finir, un gros critère requis du point de vue des développeurs noyau est que ce genre de techniques n'implique pas d'inclure dans le noyau un code qui formerait une API stable pour la programmation de drivers, chose que beaucoup, dont Linus Torvalds, ne veulent sous aucun prétexte.

COMMENTAIRE DE LECTEUR SUR LE KERNEL CORNER 91

Jean Paul Riou-Fougeras, Ingénieur systèmes et réseaux

Bonjour,

Et merci pour vos articles très intéressants.

Une petite rectification historique sur la virtualisation. Le concept LPAR développé par IBM sur ses architectures propriétaires 370/ESA est bien antérieur à 1999. Il date, pour sa commercialisation, de 1992/1993. Mais il s'agit en grande partie de l'intégration en hardware ou plus exactement dans le microcode des fonctionnalités de base de l'hyperviseur VM (*Virtual Machine*) développé et commercialisé par IBM à la fin des années 70 avec l'introduction de l'architecture 370 justement.

Dans cette architecture, les éléments techniques de la virtualisation sont déjà totalement intégrés au matériel, ce qui n'a jamais été le cas de l'architecture X86 jusqu'en 2005.

En effet, si la virtualisation au niveau processeur est toujours possible lorsque au moins 2 états de fonctionnement existent, mode superviseur/programme en 370 ou ring 0/1,2,3 en x86, cela nécessite des dispositifs particuliers pour la protection des zones hyperviseur de la mémoire centrale et de la gestion des I/O, dispositifs qui existent sur IBM/370, mais pas sur intel/x86.

De plus, cela nécessite aussi une stricte hiérarchisation des instructions machine : un processus en ring 1/user ne doit pas pouvoir exécuter une instruction ou lire des zones en mode ring 0/superviseur sans passer par un contrôle hyperviseur. C'est « très facile » à faire en IBM/370, mais vraiment tordu en intel/X86.

Pour cette raison, dans l'architecture X86, l'intégration de la virtualisation nécessite des contorsions en programmation seulement pleinement réussies par la société VMware avec son soft VMware. Il faut aussi noter que l'idée de virtualisation est issue de recherches menées par la société BULL et dans des universités françaises dans les années 60.

Cordialement.

Denis Bodor

Solutions Linux 2007, petit bilan personnel

« C'était mieux dans le temps ! » : voilà un peu le ton général de cette édition 2007. Chose amusante, c'est également ce qu'on entendait l'année précédente, ainsi qu'en 2005, en 2004, etc. Preuve qu'un certain nombre de choses restent et que tout n'était pas si différent... avant.

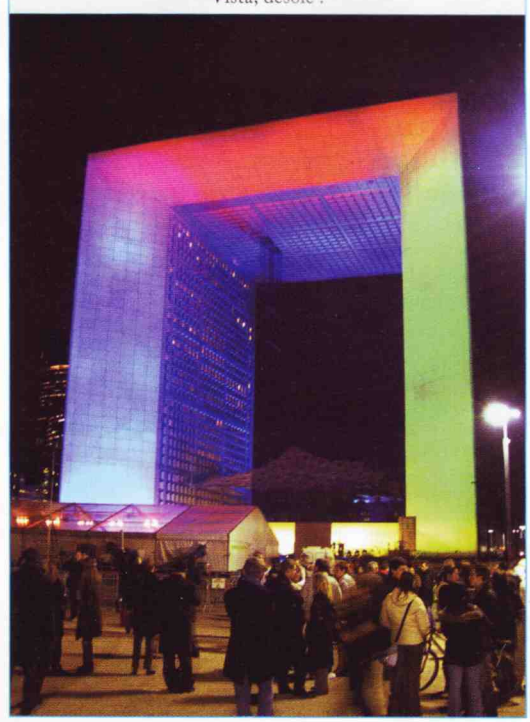
Le climat, l'ambiance et les événements d'un salon sont-ils vraiment représentatifs de l'état d'une communauté d'utilisateurs/développeurs ou d'un marché ? Personnellement, j'en doute, mais force est de constater que la prise de température lors de SL07 est relativement parlante.

La constatation première, cette année, porte sur l'aspect global du salon et la présence d'acteurs importants. Ainsi, on a pu constater qu'IBM, Apple, HP et d'autres n'avaient pas investi en temps, en énergie et en finance dans cette édition. La taille globale de l'espace exposant s'en est donc retrouvée amputée d'autant, tout en conservant les proportions associatifs/professionnels qu'on connaît d'habitude. Les plateformes *open source* et le Logiciel libre n'étant pas en déclin, c'est donc l'évènement lui-même qui semble moins intéresser les gros exposants.

Cette analyse semble se confirmer par la fréquentation de cette année. SL07 fut indéniablement marqué par une forte présence de curieux et de particuliers. Étonnant pour un salon se voulant professionnel et se tenant du mardi au jeudi. Après quelques échanges, je constate que cette impression est partagée : « il n'y a que des concurrents et des particuliers, dommage! ». Ne nous en attristons pas. Cela est sans doute signe que GNU/Linux et d'autres systèmes comme *BSD ne sont plus une alternative pour un professionnel, mais représentent un choix technologique (on le savait déjà, certes). L'alternative ou du moins la mode de l'alternative est maintenant sur le *desktop* du particulier. Le travail d'information sur les risques et le coût de la migration vers Microsoft Vista n'est sans doute pas étranger à cela. Notons au passage la coïncidence amusante entre le son et lumière proposé par la firme de Redmond devant l'Arche de la Défense et la clôture du premier jour du salon au CNIT (à quelques dizaines de mètres, donc). Les visiteurs du salon ont donc eu le plaisir de profiter du spectacle, relativement sobre, « offert » à l'occasion du lancement de Vista.

Certains penseront qu'il s'agit d'une provocation délibérée. Personnellement, je penche pour une certaine négligence ou un laxisme du service de communication de Microsoft.

Un son et lumière à la fin de la première journée du salon, merci Microsoft, mais cela ne me convaincra pas d'installer Vista, désolé !



Tarsus, repreneur de l'évènement pour cette édition 2007, a annoncé « un réel succès » avec 9842 (42 ?) visiteurs pour 208 exposants. « Chacun a pu noter que *Solutions Linux* attire, désormais, essentiellement les professionnels » est une affirmation qui me semble néanmoins s'opposer à mes constatations. Bien évidemment, l'édition 2007 reste majoritairement un évènement professionnel, mais il faut relativiser en prenant en compte les éditions précédentes. D'autre part, cette analyse est sans doute basée sur les pré-inscriptions et inscriptions à l'entrée du salon pour lesquelles un nom de société était demandé. La pertinence de cette base n'est certainement pas très importante : je me souviens avoir croisé un Seymour Cray de « Cray recherches » dans les allées ;)

Bien moins amusant et plus perfide en revanche était la prestation, dès le premier jour, d'un trio de noir vêtu, cagoulé de vert, représentant le SWATE et éructant des slogans à la gloire du Logiciel libre. « Rien d'étonnant pour *Solutions Linux* », me direz-vous. Au contraire, une petit visite sur le site du soi-disant groupe d'activistes et nous voici noyé par un raz-de-marée d'animations Flash. Surprenant pour des défenseurs du Logiciel libre et des formats ouverts. Pas dupes, ce sont finalement quelques membres de GCU-Squad (eh oui !) qui ont découvert le pot aux

roses. Les pseudo-activistes du Logiciel libre étaient des intermittents du spectacle et le nom de domaine utilisé déposé par une agence de communication. Agence qui, selon toutes vraisemblances, compte comme client une importante SSII dont le stand arborait des couleurs similaires à celle des agitateurs factices.

Il reste toutefois intéressant de noter que le « folklore » propre au domaine des développeurs, sysadmins et autres contributeurs du Logiciel libre est assimilé au point d'être gaiement repris dans une campagne de promotion. Preuve, peut-être, que le mélange d'immaturité et de haute technicité du monde open source est un fait établi et accepté.

Le stand Novell/SUSE en pleine séance de décontamination joyeuse (ou pas)

La vraie animation du salon arriva le jeudi avec l'intervention de deux agents de décontamination en combinaison jaune passant de stand en stand dans le but de nettoyer les ordinateurs souillés par du logiciel sale (propriétaire). On reconnaîtra, là, la marque du collectif GCU-Squad, soutenu dans l'opération par l'APRIL et d'autres associations défendant les valeurs du Logiciel libre. C'est ainsi que, sur le stand Novell/SUSE, le ton monta, lorsque furent évoqués les récents accords entre la société de Salt Lake City et Microsoft. Accords dont on ne connaît qu'une partie, diffusée par voie de presse. Les employés de Novell sur le stand ont, bien malheureusement pour eux, essayé de se défendre contre ce « nettoyage rigolo » relativement bon enfant. « Malheureusement », car il est relativement difficile de défendre un accord dont on ne semble pas connaître pleinement le contenu (NDA oblige). Mieux valait prendre cela avec humour et reculer comme l'ont fait d'autres, lors de précédentes éditions du salon, en se voyant gratifié d'un tee-shirt en récompense d'un excellent quota de dépôt de brevets sur les logiciels.

Solutions Linux 2007 était également le rendez-vous de quelques personnalités du paysage politique français. On a ainsi pu voir Danièle Auffray, adjointe au maire de Paris en charge des NTIC, Bernard Carayon, député du Tarn, Vincent Feltesse, secrétaire-adjoint aux NTIC du PS, Jérôme Relinger, responsable du site du PC et Francois Bayrou, président de l'UDF, qui a inauguré le salon. Il est toujours intéressant de voir ces personnalités « réaffirmer leur soutien à l'utilisation et à la conception des Logiciels libres » tout en demandant au détour d'un stand : « C'est quoi Mandriva ? ».

Toujours dans l'ambiance de la campagne présidentielle, il est important de signaler l'initiative « candidats.fr » de l'APRIL visant à regrouper les éléments intéressants en faveur (ou non) du Logiciel libre dans les programmes ou propositions des candidats à la présidence. Le site **www.candidats.fr** centralise ainsi des éléments d'information concernant plusieurs points qui peuvent être importants pour un acteur ou un utilisateur de Logiciel libre. Par exemple :

- régulation du net : quel est l'avis des candidats sur la commission administrative souhaitée par le gouvernement ?
- les risques liés aux brevets sur les logiciels ;
- l'ouverture des formats et l'interopérabilité des systèmes ;
- la DADVSI, application de la directive 2001/29CE ;
- les problèmes de ventes liées ordinateur/logiciels ;
- l'enseignement de l'informatique et les Logiciels libres dans l'éducation.

L'initiative candidats.fr propose également un questionnaire destiné aux acteurs de la campagne qui, s'ils y répondent, permettra de clairement établir la position de chacun d'eux sur les points qui nous concernent. Bien entendu, à chaque électeur ensuite d'utiliser ces éléments en accord avec ses autres centres d'intérêt et opinions politiques pour faire son choix.

Au moment où est écrit cet article, aucun candidat n'a répondu au questionnaire, même si certains ont annoncé leur volonté de le faire. Cependant, en politique, mieux vaut se baser sur des faits plutôt que sur les promesses.

Changeons radicalement de domaine avec quelques informations recueillies sur le stand Kaspersky, suite à un entretien avec J.P. Bichard. La présentation des produits a rapidement glissé vers une discussion plus ouverte sur les risques informatiques et les méthodes de développement de la société. Ainsi, mon interlocuteur a insisté sur le fait que beaucoup d'utilisateurs ne se rendaient pas compte de la source réelle des problèmes et les relations entre le code viral/malfaisant et le *spam*. Les termes de mafia, crime organisé et groupes structurés ont été lâchés à plusieurs reprises pour désigner des sociétés ou groupes mêlant recherche et développement, activités illégales, spam massif, etc. Pour Kaspersky Lab, la menace n'est clairement et définitivement plus le développeur en mal de sensations, mais des organisations criminelles fonctionnant comme des sociétés et disposant de leurs propres services de recherche.

L'autre point intéressant dans la discussion portait sur les différences culturelles existant dans le développement de logiciel. La majorité du développement sur le cœur des solutions de cette entreprise se fait en Russie par un ensemble de développeurs fonctionnant en cercle fermé. Le code résultat m'a été décrit comme extrêmement condensé et optimisé... et, par effet de bord, relativement difficile à cerner pour un développeur ne possédant pas la culture du développement russe.

Toujours à propos du code, la question de l'ouverture des sources du scanner et des éléments-clefs des différentes solutions de la société n'est pas à l'ordre du jour. Les problèmes de sécurité et la guerre entre éditeurs de solutions Anti* et les groupes développant du code viral est l'une des raisons évoquées. Enfin, pour clore le sujet, la raison majeure de la non-prolifération de code viral dans les systèmes comme GNU/Linux ou *BSD reste principalement la faible part de marché de ces systèmes, pour Kaspersky Lab. Même si cela joue un rôle important, je reste toutefois convaincu que la qualité du code issu du développement open source est un point à ne pas négliger, même s'il est vrai que, pour une machine desktop, la cible reste la juste émulsion entre le système le plus populaire et le plus sensible aux attaques.

Plus rafraîchissant, un petit morceau de stand dans l'espace associatif était consacré à un projet qui m'a littéralement « tapé dans l'œil » (et je ne suis pas le seul) : Tux Droid. Développé par une société belge de manière extrêmement ouverte, le Tux Droid est un périphérique autonome en forme de Tux (on s'en doutait !). Celui-ci intègre un grand nombre de fonctionnalités parmi lesquelles :

- il est animé et peut, à distance, gesticuler ses ailes, ses paupières et son bec ;
- il peut tourner sur lui-même ;
- il peut diffuser des sons et de la musique via un *stream* audio ;
- il dispose d'un micro et peut enregistrer également via un mécanisme de streaming ;
- il dispose d'un émetteur/récepteur infrarouge lui permettant de contrôler des appareils Hi-fi et de communiquer avec d'autres Tux Droid ;
- il détecte le niveau de luminosité ambiante ;
- il dispose de détecteurs permettant une certaine interactivité.

Tout cela, bien sûr, étant contrôlé depuis un ordinateur sous GNU/Linux (ou autre) via un adaptateur USB en forme de poisson. Le périphérique est conçu autour d'un microcontrôleur Atmel AVR ATMEGA88 ou ATMEGA48 (RF) dont le firmware est disponible, ouvert et documenté. Plus qu'un simple jouet disposant d'applications pratiques, Tux Droid est aussi une plate-forme de développement. Les futurs utilisateurs sont invités à participer au développement, installer leur chaîne de compilation et proposer de nouveaux firmwares.

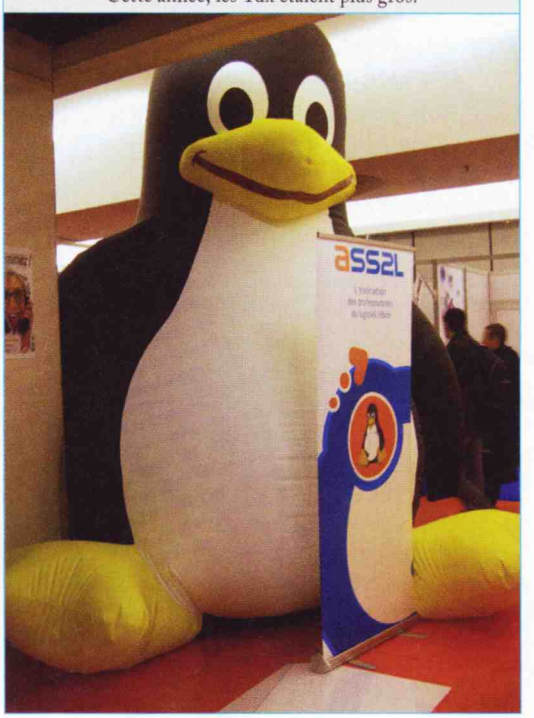
Côté ordinateur, le contrôle du Tux Droid est assuré par un ensemble de démons avec lesquels il est possible de dialoguer facilement. Un *binding* Python permet de facilement faire ses premiers pas après avoir essayé le client fourni. On choisira ensuite de pousser ou non dans le développement en se tournant vers un langage comme C ou C++.

Le site http://www.tuxisalive.com regroupe toutes les informations utiles concernant le périphérique et le développement. Ceci incluant les sources du *firmware*, mais aussi le schéma électronique complet (dans le SVN). Pour l'heure, la production en série n'est pas pleinement démarrée, mais il est déjà possible de passer des pré-commandes. Comme je sens la question poindre le bout de son nez, sachez que Tux Droid devrait être commercialisé à 79 euros (pour le Tux Droid, avec batterie rechargeable, adaptateur secteur, *dongle* Wireless USB et câbles).

Je ne vous cache pas que le mien est déjà pré-commandé et qu'il est fort probable que l'on reparle de cette bestiole dans les pages du magazine dans les mois qui suivent.

Conclusion, ce salon était fort intéressant et loin d'être une perte de temps pour le visiteur curieux. Côté professionnel, comme à l'habitude, il s'agissait surtout d'assurer sa présence sur le marché et éventuellement de démarcher des clients. Bref, une bonne édition sans pour autant être exceptionnelle.

Cette année, les Tux étaient plus gros.



Denis Bodor,

db@ed-diamond.com

lefinnois@lefinnois.net

Philippe Bruhat

OSCON Europe 2006

Du 18 au 21 septembre 2006 s'est tenue à Bruxelles la deuxième édition européenne d'OSCON. OSCON, pour Open Source CONference est une conférence annuelle organisée par O'Reilly Media aux États-Unis depuis 1997, et en Europe depuis 2005. La convention était hébergée à l'hôtel Le Plaza, un des luxueux hôtels au nord de la ville.

La conférence avant la conférence

Avant même le début de cette conférence (professionnelle s'il en est), une autre conférence avait lieu : EuroFOO (FOO signifiant ici *Friends Of O'Reilly*). C'est une conférence informelle, sur invitation seulement, où Tim O'Reilly invite tout un tas de gens intelligents et à la pointe de l'innovation (les *alpha geeks*, comme il les surnomme parfois), pour essayer de voir dans quelle direction la technologie avance.

Plusieurs des invités d'*EuroFOO Camp* participaient également à OSCON, comme on a pu le voir aux nombreux T-shirts orange décorés du logo de l'événement.

Présentation

Comme le montre l'emploi du temps de la conférence (accessible à l'adresse **http://conferences.oreillynet.com/euos2006/grid/**), il n'y avait pas de quoi chômer : des présentations de 9h à 17h40, en parallèle dans 5 salles pendant les 3 jours, des expositions après 19h, sans parler des couloirs pleins de monde en train d'échanger sur tous les sujets possibles.

Cette année, le thème de la conférence est *Opening Innovation*.

Jour 0 – Les tutoriels

Le lundi est consacré aux séances de tutoriels ; mais avec quatre tutoriels en parallèle pendant chaque demi-journée, il était évidemment impossible d'assister à tout ! Il en sera d'ailleurs de même pour toute la durée de la conférence. Ce compte-rendu va donc vous donner une vue partielle de la conférence, et essayer de rendre l'ambiance et la diversité des sujets abordés.

Ruby on Rails – Marcel Molina

Les tutoriels commencent dès 8h30 du matin, avec *Jabber Boot Camp*, par Peter Saint-André et Ralph Meijer, *Start-ups 2.0*, par Marc Hedlund, *Data Warehousing with MySQL*, par John Paul Ashenfelter et *Ruby on Rails* par Marcel Molina.

N'ayant pas vraiment de connaissance de RoR (Ruby on Rails), mais n'arrêtant pas d'en entendre parler, j'ai voulu voir ce qu'il en était.

Marcel Molina sait de quoi il parle, puisqu'il fait partie de 37signals, la boîte qui a publié Ruby on Rails (inspiré de leur logiciel maison *BaseCamp*). Ruby on Rails est un *framework* web très à la mode en ce moment.

Comme l'a fait remarqué Piers Cawley, tout le bruit actuel autour de Ruby on Rails, c'est un peu « *a bunch of Java programmers finally discovering how cool Perl is* » (un tas de programmeurs Java qui découvrent finalement à quel point Perl est cool).

Marcel rappelle le principe du MVC (*Model-View-Controler*, issu de SmallTalk, sur lequel est basé RoR), et montre par l'exemple les deux principes de RoR : DRY (*Don't Repeat Yourself*) et « *Convention over Configuration* ».

Ainsi avec ActiveRecord, l'ORM de RoR (*Object Relational Mapper*, un ensemble de classes qui gèrent la persistance des données dans une base de données en évitant au programmeur de devoir toucher à la base de données), il n'y a besoin de définir les attributs de la classe qu'en un seul endroit. Ces attributs seront également les noms des colonnes dans la table qui portera le même nom que la classe. On le voit, les *rubyistes* eux aussi pensent que la paresse est une vertu.

RoR insiste également beaucoup sur les tests et la documentation, en créant en particulier tous les fichiers et le cadre nécessaire pour ceux-ci. Bon, c'est quand même à vous de les écrire à la fin...

Il présente enfin le système de « migrations » d'ActiveRecord. C'est un système qui permet de mettre à jour les modifications de sa base de données sans s'arracher les cheveux. Le principe est qu'ActiveRecord va écrire pour vous les scripts de migration (d'où le nom) de la base de données au fur et à mesure de l'évolution de votre base de code. On définit les modifications de la base d'une façon abstraite, et le SQL nécessaire pour faire évoluer le schéma de la base est généré par Ruby, pour le moteur de base de données cible.

Damian Conway – The 7 Principles of Better API Design

L'après-midi propose quatre nouveaux tutoriels : The *7 Principles of Better API Design* par Damian Conway, *Building Better Ajax Applications* par Alex Russell, un *Workshop on Open Source Licensing for Businesses* par Martin von Haller Groenbaek et *Building Passionate Users* par Kathy Sierra.

Damian Conway est l'un des meilleurs présentateurs de la communauté Perl, je n'ai donc pas manqué son tutoriel.

Les 7 principes sont les suivants (et il les illustre tous avec des modules qu'il a écrits) :

Damian Conway
Photo James Duncan Davidson/O'Reilly Media

1. *Do one thing really well*

Un module devrait ne faire qu'une chose, mais très bien.

`Perl6::say` exporte seulement la fonction `say()`, qui fonctionne comme en Perl 6 (en rajoutant un `\n` au bout).

2. *Design by coding*

Écrire le code qui va utiliser le module que l'on est en train de concevoir est une très bonne manière d'améliorer sa conception. Le développement piloté par les tests (TDD, *Test Driven Development*) est une manière d'y arriver, les scripts de test utilisant la bibliothèque qu'ils doivent tester. Demander à ses futurs utilisateurs de proposer l'interface qui leur plairait en est une autre.

`Regexp::Common` a ainsi l'interface d'un *hash*, parce que c'est la plus logique pour les utilisateurs du module.

3. *Evolve by substraction*

L'évolution de l'interface se fait en déplaçant la complexité vers les fonctions de bas niveau, donc en soustrayant de la complexité des interfaces de haut niveau (celles que voit l'utilisateur).

`IO::Prompt` n'exporte qu'une seule fonction, dont le comportement (qui peut être assez complexe) est entièrement guidé par les options d'appel de cette fonction.

4. *Declarative beats imperative*

Il est préférable d'avoir une interface déclarative, c'est-à-dire où l'utilisateur décrit ce qu'il veut faire, plutôt qu'impérative, où il décrit comment le faire.

`Getopt::Euclid` utilise la documentation du programme pour écrire l'analyseur d'options de ligne de commande.

5. *Preserve the metadata*

On ne devrait jamais devoir répéter deux fois la même chose à un programme. S'il apprend quelque chose à un moment donné, il devrait conserver cette information pour plus tard.

`Config::Std` conserve les commentaires des fichiers de configuration qu'il lit pour pouvoir les laisser en place lors de la mise à jour du fichier.

6. *Leverage the familiar*

L'interface la plus simple est celle que les utilisateurs connaissent déjà.

`Log::StdLog` permet d'utiliser l'interface de `print()` pour gérer divers niveaux de log.

7. *The best code is no code at all*

Le simple fait d'écrire le `use` devrait suffire à invoquer la magie du module.

`Object::Dumper` permet de faire un `print()` d'un objet complexe et de voir non pas son adresse, mais un *dump* de l'objet.

La présentation a beaucoup influencé mes deux voisins... Aaron Crane a écrit la première version de `Acme::InsultingErrors`, qui ajoute la chaîne « `, you idiot` » (avec une liste de synonymes digne d'`Acme::MetaSyntactic`) à la fin des messages d'erreur et avertissements, tandis que Jouke Visser a modifié son dernier module, `OA`, pour qu'il utilise `Perl6::Export::Attrs` au lieu de `Perl6::Export`, comme Damian le recommande désormais.

Jour 1 – Bienvenue !

La véritable *Open Source Convention* commence ce mardi avec les *keynotes*, pendant lesquelles nous verrons se succéder :

Scene setting and Introduction – Nathan Torkington, Allison Randal et Nikolaj Nyholm

En fait, seuls Nat et Allison nous accueillent : Nikolaj Nyholm, le troisième *program chair* est absent pour cause de naissance de son fils.

Nat nous explique le *business model* d'O'Reilly : en fait, O'Reilly vend des « mouvements populaires ». Selon ce business model, il s'agit d'identifier ces mouvements, de trouver ce dont les gens ont besoin en rapport avec ces mouvements et de leur vendre.

Les gens ont besoin d'apprendre, de se documenter, de se rencontrer. Les livres et les conférences ne sont que des effets secondaires de la popularité de ces mouvements, grâce auxquels la société O'Reilly fait de l'argent.

L'*Open Source* fait partie de ces « mouvements populaires ». Il existe d'autres mouvements qui s'appuient sur les mêmes principes.

En plus du logiciel, on peut « open sourcer » la culture, l'industrie, la démocratie, l'économie. Ainsi *Creative Commons* « open source » la culture, les *makers* (les bidouilleurs inspirés de *MAKE Magazine*) « open

sourcent » les objets, *TheyWorkForYou.com* « open sourcent » la démocratie.

Au-delà du logiciel, de la culture, de l'industrie, le but de cette conférence est de savoir ce que nous pouvons chacun apprendre les uns des autres.

Open Source 2.0 – Tim O'Reilly

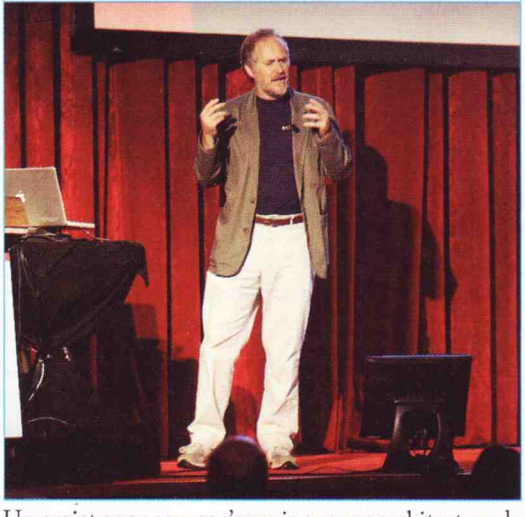
Si le titre donnait l'impression d'une variation sur le *buzzword* à la mode lancé par Tim O'Reilly lui-même (le web 2.0), ce n'est pas le cas : Tim présente ici ses inquiétudes et ses prédictions pour l'avenir.

Il rappelle ce que fait O'Reilly : changer le monde en diffusant le savoir des innovateurs. Avec les *bloggeurs* du « radar » O'Reilly, ils essayent de détecter ce qui change.

Sa présentation comporte plusieurs grands points :

1. Le web 2.0, c'est une architecture de participation, au-delà du développement logiciel.

Tim O'Reilly
Photo James Duncan Davidson/O'Reilly Media



Un projet open source s'appuie sur une architecture de participation. Cependant, un projet comme Wikipedia, qui ne concerne pas un logiciel, mais le savoir, a la même architecture.

Le web 2.0, ce sont des systèmes qui exploitent les effets de réseau pour s'améliorer avec le nombre de leurs utilisateurs. Idéalement, cela se fait par une architecture dans laquelle les gens construisent le système en s'occupant simplement de leurs propres centres d'intérêt.

Le web 2.0, ce sont donc des logiciels dont les utilisateurs sont des composants. Il utilise les mêmes leviers que les projets open source, mais il y a aussi de nombreux défis, en particulier, le fait que le logiciel n'a plus besoin d'être diffusé : on utilise le web.

Ce qui fait que :

2. Les licences open source sont obsolètes.

Google peut typiquement utiliser n'importe quel logiciel ou licence, mais il ne donne jamais du logiciel en retour, seulement du service.

Pour Richard Stallman, il n'y a pas de problème tant que l'ordinateur qui fait marcher le logiciel n'est pas celui de l'utilisateur.

Selon Tim, il faut réinventer l'Open Source pour un monde dans lequel le logiciel est exécuté et pas diffusé, où les applications s'appuient sur de gigantesques bases de données plus que sur du code et où ces applications sont mises à jour en permanence.

Sa question est donc : quelles sont les licences adaptées pour les *web services* ? Que serait un web service Open (Source) ?

Il est intéressant de constater que les services propriétaires donnent le framework. Ainsi, Basecamp (http://www.basecamphq.com/) est l'application produite par 37signals (http://www.37signals.com/). Le framework utilisé pour produire le site est Ruby On Rails, qu'ils ont rendu disponible avec une licence open source.

Dabble DB (http://dabbledb.com/) s'appuie sur Seaside (http://www.seaside.st/), un framework en SmallTalk, lui aussi disponible avec une licence libre.

En résumé, les applications propriétaires donnent des API ouvertes, et les services propriétaires donnent des frameworks ouverts.

3. La compétition n'est pas symétrique.

Craigslist (http://www.craigslist.org/) n'a que 18 employés, et c'est pourtant l'un des sites les plus visités sur le web.

4. L'hébergement est un avantage.

Ainsi que l'a dit Debra Chrapaty, vice-présidente de *Windows Live Operations* : « *In the future, being on someones's platform will mean being hosted on their infrastructure.* » (Dans l'avenir, utiliser la plate-forme de quelqu'un signifiera être hébergé sur leur infrastructure).

Le mouvement Open Source sera donc confronté à un problème d'infrastructure.

Il faudra donc s'intéresser de plus près aux outils qui facilitent la gestion d'infrastructures de grandes dimensions : Nagios (une solution de *monitoring*, http://nagios.org/), Memcached (un cache mémoire distribué, http://www.danga.com/memcached/), Hadoop (un système de fichiers distribué en Java, http://lucene.apache.org/hadoop/), OpenID (un système d'authentification ouvert, décentralisé et gratuit, http://openid.net/).

5. Qui possède les données ?

Si vous ne pouvez pas déplacer vos données, c'est qu'elles ne sont pas à vous.

Il renvoie en particulier aux sites de http://movemydata.org/ http://microformats.org/ et bien sûr http://creativecommons.org/.

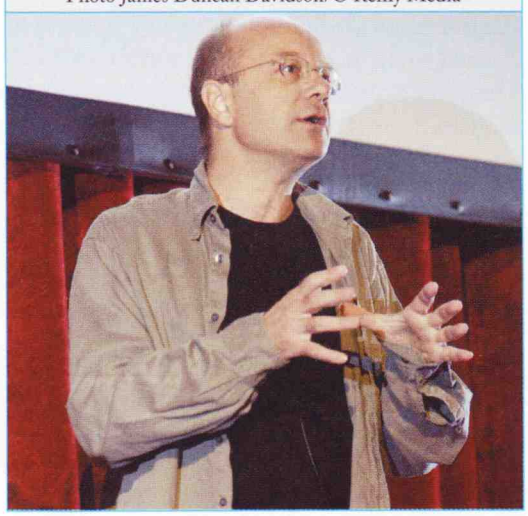
Ainsi, nous ne savons pas encore :

- ce que sera la plate-forme du futur ;
- quels seront les débouchés commerciaux créés par les systèmes qui s'améliorent eux-mêmes ;
- ce qui créera la compétition et les clientèles captives ;
- ce que cela signifie pour un service web d'être ouvert ;
- ce que cela signifie pour les données d'être vraiment libres.

En guise de conclusion, Tim cite Ray Kurzweil : « *I'm an inventor, and I started looking at long-term trends because an invention has to make sense in the world in which it was finished, not the world in which it started.* » (Je suis un inventeur, et j'ai commencé à m'intéresser aux tendances à long terme parce qu'une invention doit avoir une signification dans le monde où elle sera terminée, pas dans le monde où elle a été commencée.)

Attention, Please! Who Are We? – Tor Nørretranders

Tor Nørretranders
Photo James Duncan Davidson/O'Reilly Media



Tor commence sa keynote en parlant du jeu de l'ultimatum. Étant donné une somme d'argent et deux joueurs, le premier joueur propose un partage. Si le second l'accepte, le partage est fait selon les exigences du premier, sinon personne ne reçoit d'argent.

À partir de cet exemple de coopération forcée, Tor explique que c'est la coopération qui nous fait « briller ». D'autres expériences montrent que ce n'est pas le cas de la coopération avec un ordinateur. Nous avons donc besoin d'êtres humains pour coopérer. La question serait donc « pourquoi les êtres humains veulent-ils coopérer entre eux ? ».

Il commence par mentionner Darwin, et résume sa théorie de l'évolution à « la vie, c'est un problème de survie ». Pour l'anecdote, Tor rappelle que Darwin était exaspéré par les paons. Qu'un animal se rende aussi voyant (pour ses prédateurs) ne rentre pas du tout dans la théorie de l'évolution vue comme « la survie du plus adapté » !

Attirer l'attention peut vous procurer des rapport sexuels, mais c'est aussi dangereux (au moins pour le paon). Selon la théorie qu'explique Tor, si le paon a une ornementation aussi voyante, ce serait en vertu du principe du handicap : pour montrer qu'on est riche, on gaspille ; pour montrer qu'on est fort, on prend des risques. Les êtres humains montrent qu'ils ont de vastes ressources en faisant montre d'altruisme, en étant généreux, en prenant des risques. Le business model serait : oser prendre des risques et partager nos richesses attire l'attention, ce qui a pour retombées le sexe, le travail et la reconnaissance. Le sexe serait à l'origine de tout ce qui est noble !

Tor Nørretranders est probablement l'intervenant qui a eu le plus de succès. Suite à cette keynote, il a d'ailleurs donné une autre présentation (hors programme) qui fait salle plus que comble, avec des auditeurs assis par terre sur plusieurs rangs.

Le livre de Tor Nørretranders et Jonathan Sydenham, *The Generous Man: How Helping Others Is the Sexiest Thing You Can Do*, n'a pas été traduit en français d'après mes recherches.

Les premières sessions

J'ai manqué les sessions suivantes, trop occupé par mes retrouvailles avec quelques membres de la communauté Perl et par une interview de Tim O'Reilly réalisée conjointement avec un journaliste du magazine *IT Professional*.

Si j'y avais assisté, j'aurais pu vous parler de Drupal, des microformats, de Google code, d'évangélisation Open Source, etc. (voir l'emploi du temps).

J'ai tout de même pu assister à la fin de la session *Gender in Open-Source*, par Bernard Krieger. Il présentait ses conclusions sur une étude basée sur un sondage concernant le rôle des femmes dans les projets open source. Hélas, les questions posées dans le sondage, sa propre analyse et même sa manière de gérer l'échange avec le public très féminisé (des femmes qui étaient très conscientes de leur position dans la communauté) montraient combien il était lui-même englué dans ses propres stéréotypes et préjugés. Pour rattraper, je vous donne quelques sites sur le sujet :

- La liste de diffusion **women@apache.org** de la fondation Apache. Archives : **http://mail-archives.apache.org/mod_mbox/www-women/**
- Debian Women : **http://women.debian.org/**
- LinuxChix : **http://linuxchix.org/**
- grep|grrl : **http://grepgrrl.org/**

Open Data AWOL – Steve Coast

L'après-midi commence avec une nouvelle série de keynotes.

Steve Coast, d'OpenStreetMap (**http://www.openstreetmap.org/**) expose le problème du manque de données libres, pour tout ce qui va des données géographiques jusqu'aux annuaires téléphoniques.

Avec l'avènement du Web 2.0, ceci pose de plus en plus de problèmes aux petites sociétés qui se montent, car elles ne disposent pas des données utilisables librement pour leurs sites.

Journalism via Computer Programming – Adrian Holovaty

Cette seconde keynote présente l'idée de journalisme assisté par ordinateur. Il s'agit de reprendre les données brutes utilisées par les journalistes traditionnels, et de les rendre disponibles pour tous. Cela permet des utilisations variées et créatives, comme :

- *Faces of the Fallen* (http://projects.washingtonpost.com/fallen/), hébergé par le *Washington Post* qui présente les données concernant tous les soldats américains morts en Irak et en Afghanistan.
- *Chicago Crimes* (http://www.chicagocrime.org/), qui rend accessible la base de données des crimes commis à Chicago avec un moteur de recherche, une géolocalisation et même la possibilité de tracer son chemin sur une carte, et de voir les crimes qui ont été commis aux alentours.
- *The US Congress Votes Database* (http://projects.washingtonpost.com/congress/), encore hébergée par le *Washington Post*, qui permet de visualiser *tous* les votes du Congrès des États-Unis depuis 1991. Là encore, on peut visualiser l'information selon de nombreux critères (par vote, par votant, par parti), visualiser les votes d'un membre du Congrès par rapport à l'opinion des partis démocrate et républicain, et même voir le décompte des votes par signe astrologique des votants !

Imaginez les *feeds* RSS que cela rend possible !

Bref, l'idée est de récupérer les données brutes, et de faire des trucs cools avec. Si l'Open Source c'est de rendre le code disponible, alors l'*Open Journalism*, c'est de rendre les données disponibles. Dans les deux cas, on facilite la compréhension par la transparence et on encourage les travaux dérivés.

Quelques sessions de l'après-midi

Parmi les nombreuses sessions de l'après-midi, Google était assez représenté avec *The Google Data API*, par Frank Mantek, *Rolling Your Own Google Maps*, par Scott Davis et une session *Google and Open Source* dans la salle dédiée aux sponsors.

Dans *An Economic Interpretation of the Evolution of the Free/Open Source Software*, Lorenzo Benussi présente l'enchaînement des différentes époques du développement de la communauté FLOSS. En partant du *New thinking* de Vanevar Bush (1945), et du projet MAC (*Mathematics and Computation*) du MIT, en passant par Unix, l'ordinateur personnel, le projet GNU pour arriver à Linux, il montre l'importance de la communication comme moteur et comme moyen du développement du *Free/Libre/Open Source Software*.

Dans *Making It Work: How to Build a Successful Open Source Project*, Louis Suarez-Potts présente comment réussir son projet open source. Fort de son expérience personnelle avec OpenOffice.org, il aborde de nombreuses questions, dont la différence entre les projets « organiques » (qui grossissent à partir du travail de quelques développeurs) et les projets sponsorisés (par des entreprises) ; le problème de la licence et du copyright (on peut toujours changer la licence, mais il faut bien s'assurer du copyright, sinon chaque contributeur conserve le copyright des bouts de code auxquels il a contribué) ; l'intérêt de créer une gratification immédiate (par exemple avec un *wiki*) ; le besoin de transparence sur les procédures de revue, d'acceptation ou de rejet du code des contributeurs ; le besoin de communication, de marketing, de promotion ; la nécessité de créer une communauté.

Selon lui, pour réussir son projet open source, il faut :

- démarrer en petit groupe ;
- voir grand ;
- être ouvert au changement ;
- avoir des procédures ;
- faire du marketing ;
- savoir quoi répondre aux questions comme : « pourquoi nous rejoindre ? », « comment ? », « pourquoi l'utiliser ? », « pourquoi avoir fait ce projet en open source ? ».

Il conclut en expliquant ce que fait OOo désormais : faciliter les contributions, utiliser les extensions (comme FireFox), avoir des développeurs régionaux, communiquer en direction des entreprises, des écoles, des gouvernements, des individus.

Au final, le but est de permettre à tout le monde de contribuer à OpenOffice.org, par des tests, du code, de nouvelles extensions, des traductions, du graphisme, de la documentation, du soutien enthousiaste (ou en payant une bière aux développeurs).

La journée se termine dans le hall d'exposition, où les sponsors accueillent les participants autour de rafraîchissements pour faire la démonstration de leurs produits et de leur savoir-faire.

Jour 2

La journée commence par une nouvelle session de keynotes :

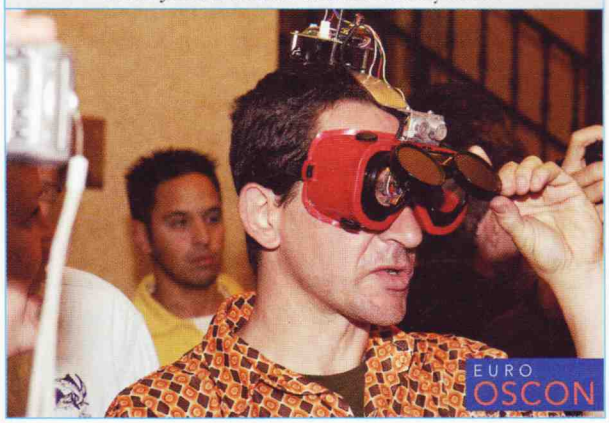
- *Welcome honestly.gov*, dans laquelle Tom Steinberg présente un nouveau type d'organisation à but civique ou d'amélioration de la société comme mySociety.org (http://www.mysociety.org/) qui crée des sites web pour améliorer la vie quotidienne des gens dans la société, WriteToThem.com (http://www.writetothem.com/), qui permet aux citoyens britanniques d'entrer en contacts avec leurs représentants dans les institutions de l'État, TheyWorkForYou.com (http://www.theyworkforyou.com/), qui couvre les débats des diverses assemblées représentatives, ou PledgeBank (http://www.pledgebank.com/), qui recueille des promesses sous la forme « Je le ferai, mais à condition que vous m'aidiez » (apparemment, cela se traduit en français par « promessothèque »).

Quand on regarde un peu ces sites, on se demande s'il en existe du même genre pour la France, et on se dit que s'ils n'existent pas, alors il est urgent de les créer !

- Dans sa présentation *Make and the Re-emergence of DIY Tech*, Dale Dougherty nous présente le magazine *MAKE*, qu'il dirige en présentant en vrac : des photos prises depuis un cerf-volant, un lance-flamme fait maison, des vidéos prises par une caméra vidéo accrochée à une fusée artisanale, un scooter à énergie solaire, le *crafter manifesto*, un caddie à moteur et des espèces de jouets à manivelle (*hand-cranks*), tous réalisés par des *makers*.

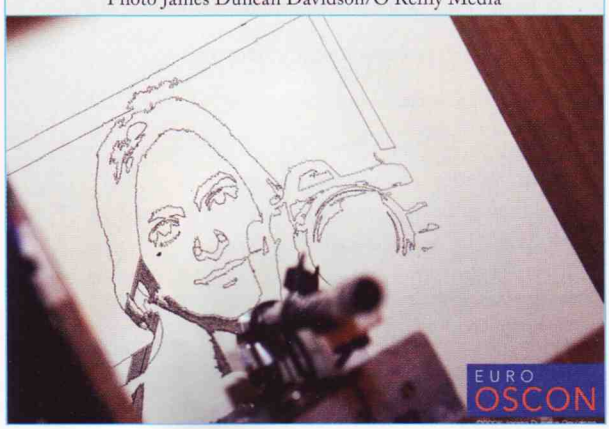
Les makers (terme qu'on pourrait traduire par « bricoleurs » ou « bidouilleurs d'objets ») fabriquent vraiment des trucs incroyables !

Les makers et leurs drôles de machines
Photo James Duncan Davidson/O'Reilly Media



La présentation *JavaScript: It's Happening All Over Again!* de James A. Duncan et Mark Fowler était sponsorisée par Zimki, une boîte qui vend du service basé sur JavaScript côté serveur. On est dans une logique de plate-forme, comme décrite par Tim O'Reilly au début de ce compte-rendu. Mark et James étant des piliers de la communauté Perl, la salle est pleine de hackers Perl, bien que le sujet soit 100% JavaScript !

Le Drawbot de Bre Pettis
Photo James Duncan Davidson/O'Reilly Media



Après les dernières présentations, ont lieu les BOF sessions. Je me retrouve donc dans une grande salle avec 5 ou 6 autres personnes pour parler d'Act (http://act.mongueurs.net/), le logiciel de gestion de conférences utilisé depuis près de 3 ans par la communauté Perl européenne. Des utilisateurs (ou développeurs ?) de Pentabarf (http://pentabarf.org/) sont également présents. La liste de fonctionnalités souhaitées se trouve encore augmentée...

La soirée sera occupée par la *MAKE Fest* où nous verrons les makers invités présenter leurs travaux. Parmi les plus intéressants, on peut citer un robot qui dessine au feutre d'après une image en noir et blanc, ou un PC transformé en machine à faire le café.

Jour 3 – C'est déjà fini !

On commence avec les dernières keynotes de cette année :

New Innovation Models, Policy-making and Lobbying – Florian Mueller

Florian Mueller est le fondateur de NoSoftwarePatents.com (http://www.nosoftwarepatents.com/. Il explique qu'un nouveau combat est en cours : les nouveaux modèles d'innovation créent de nouvelles difficultés. En vrac, on peut citer : les brevets logiciels, les nouvelles règles du copyright, le *fair use*, le DRM, la responsabilité des auteurs de logiciels, des modérateurs de forum, des webmasters.

Il expose aussi quelques réussites du lobbying auprès du Parlement européen.

Architecting Babel – Robert « r0ml » Lefkowitz

r0ml vient parler de localisation (au sens de traduction d'application). Il rappelle que la langue la plus parlée au monde est le chinois, et que 90% des gens ne parlent pas anglais.

Avoir le code source d'un logiciel, c'est bien, mais encore faut-il avoir des identificateurs et des commentaires compréhensibles. En fait, pour aller plus loin dans la localisation (et donc l'accès aux logiciels), il explique qu'il ne faut pas se contenter de traduire les applications, il faut aussi traduire le code source : les identificateurs, les commentaires, les mots-clés.

Il réveille les horribles souvenirs des développeurs qui ont eu à manipuler les versions d'Excel dont le langage de macro était traduit (et évidemment incompatible entre deux versions d'Excel dans des langues différentes).

Cette keynote a certainement touché un point sensible, si on en juge par la quantité d'interventions du public.

La dernière demi-journée

Ayant détecté un trou dans l'emploi du temps, des perleurs tuyautés par Nat Torkington ont réussi à récupérer une salle pour une scéance de *lightning talks*. J'ai eu l'occasion de découvrir un format que je ne connaissais pas, le *twenty twenty* : il s'agit d'un enchaînement de 20 diapositives de 20 secondes chacune. L'avantage est de libérer le présentateur de son portable.

Piers Cawley explique comment se faire prendre en photo. J'expose les photos compromettantes de YAPC Europe 2005. Dennis Kaarsemaker présente Upstart (un remplaçant d'`init`, voir http://upstart.ubuntu.com/), Suw Charman présente l'*Open Rights Group* (un genre d'EFF avec moins d'avocats, http://www.openrightsgroup.org/), Ewan Spence nous parle de *podcasting* (il a d'ailleurs enregistré les lightning talks), Russ Nelson présente un clavier *bluetooth* adapté à la forme de la main (et utilisable d'une seule main), Damian Conway nous parle de 101 choses qu'il aime dans Perl 6 (101 diapos en 5 minutes !), Gervase Markham parle du *phishing* et Jesse Vincent (qui a rédigé ses diapos pendant les présentations précédentes) parle de Jifty.

Pour une session improvisée, ce fut plutôt réussi.

Je retrouve ensuite le cours normal de la conférence, et assiste à une présentation plus longue de Damian Conway (*The Conway Channel 2006*), où il présente d'une manière toujours aussi éblouissante deux des modules qu'il a créés cette année : `List::Maker` (ou comment abuser de `<>` pour forcer une syntaxe Perl 6 sur Perl 5) et `Contextual::Return` (qui permet d'écrire des fonctions qui renvoient des valeurs différentes en fonction du contexte dans lequel elles sont employées).

La dernière présentation à laquelle j'ai assisté est celle de Brad Neuberg, qui parle de *Douglas Engelbart's HyperScope: Taking Web Collaboration to the Next Level Using Ajax and Dojo*.

Douglas Engelbart a inventé beaucoup de choses : la souris, l'hypertexte, l'email, les systèmes fenêtrés. Brad résume cela par cette phrase : « Edison a conçu (*designed*) la première moitié du XXe siècle, et Engelbart la seconde moitié.

Comment le laboratoire de Stanford, où il travaillait, a-t-il été aussi prolifique ? Grâce à l'*augmentation*. Cette idée est présentée dans un article publié par Engelbart en 1962 : *Augmenting human intellect*.

L'augmentation dans ce cas, ce n'est pas une amélioration à la Matrix, c'est une évolution accélérée. Il s'agit d'accélérer le processus d'amélioration. L'homme a disposé de plusieurs outils pour exercer sa pensée : le langage (depuis 40 000 ans ?), l'écriture (depuis 3 500 à 6 000 ans) et, depuis les années 50, l'ordinateur.

Ce ne sont que des outils, et on les maîtrise mieux avec une formation, de l'entraînement et une méthodologie. (S'il n'y a pas de formation, c'est qu'elle est implicite.) L'idée d'Engelbart était de cibler les capacités principales de l'humain et de les améliorer encore.

L'outil pour y arriver était HyperScope. Le projet de Brad Neuberg est une mise à disposition d'HyperScope sur le web, grâce à une application en JavaScript disponible sur http://hyperscope.org/.

Après le déjeuner, la conférence se termine sur une dernière keynote, par Mark Shuttleworth, fondateur d'Ubuntu, qui présente les obstacles qui attendent l'Open Source.

Bilan

Tim O'Reilly définit le business model de sa société comme étant de « diffuser le savoir des innovateurs à travers ses livres, services en ligne, magazines et conférences ». À en juger par le nombre et la diversité des conférences dans lesquelles O'Reilly Media est impliquée pour 2007, c'est clairement un marché sur lequel la société s'investit de plus en plus : de 4 conférences annuelles entre 2001 et 2004, on passe à 6 en 2005, 7 en 2006 et O'Reilly liste déjà 11 conférences sur http://conferences.oreillynet.com/ pour 2007.

Les sujets vont d'ailleurs bien au-delà de l'informatique :

- *Emerging Telephony Conference* ;
- *Emerging Technology Conference* ;
- *Web 2.0 Expo* ;
- *MySQL Conference and Expo* ;
- *RailsConf* ;
- *Where 2.0 Conference 2007* ;
- *O'Reilly TOC Conference* ;
- *Ubuntu Live Conference* ;
- *O'Reilly Open Source Convention* ;
- *O'Reilly Energy Innovation Conference* ;
- *RailsConf Europe*.

En attendant, si la tendance d'OSCON Europe à descendre vers le sud continue (Amsterdam en 2005, Bruxelles en 2006), on peut raisonnablement espérer voir OSCON Europe à Paris en 2007. :-)

Philippe « BooK » Bruhat

book@mongueurs.net,
de Lyon.pm et Paris.pm.

Philippe Bruhat, Sébastien Aperghis-Tramoni

Brèves de Perl

Si vous faites partie de ceux et celles qui pensent qu'on ne parle pas assez de Perl dans GNU/Linux Magazine France, cette rubrique devrait vous plaire. ;-) Nous allons vous présenter tous les mois deux ou trois nouvelles marquantes de la communauté Perl, sous forme de « brèves ». Voici donc les brèves Perl de mars 2007.

Plat_Forms

Alimenter (ou au contraire faire taire) les *trolls*^Wdiscussions pour savoir quel est le meilleur *framework* web et dans quel langage, c'est un peu l'idée du concours *Plat_Forms*, dont le but est de mettre en compétition des équipes de programmeurs de haut niveau pour réaliser un site web en utilisant le langage et le framework que chacune représente, dans un temps limité. Cette première édition s'est déroulée du 25 au 26 janvier à Nuremberg, en Allemagne. Les équipes disposaient donc de 30 heures pour développer un site en respectant le cahier des charges fixé par le jury, un portail communautaire où les utilisateurs peuvent se contacter en fonction de leur tempérament.

Neuf équipes de trois personnes concourraient, trois pour chacun des langages défendus : Perl, PHP et Java. Curieusement, Python, Ruby et .Net, malgré le bruit autour de leurs plateformes logicielles respectives Django et RoR, n'ont pas attiré suffisamment d'équipes pour être représentés.

Les trois équipes Perl étaient :

- PlusW (Allemagne), avec Mason (présenté dans ce numéro de GLMF) ;
- Revolution Systems (Kansas, USA), avec leur propre MVC, Gantry ;
- l'État de Genève et Optaros (Suisse), avec Catalyst.

On peut noter la présence dans cette dernière équipe de deux Mongueurs genevois qui sont déjà venus aux conférences Perl en France, Laurent Dami et Cédric Bouvier.

La phase de développement s'est bien sûr achevée le 26 janvier, et les équipes ont, toutes, pu rendre leur projet dans les temps. Ils vont ensuite être évalués par un jury composé de représentants des différents langages. Les résultats ne sont pas attendus avant mai, mais le site de Plat_Forms indique que le but n'est pas de désigner un vainqueur, mais plutôt d'évaluer impartialement les points forts et faibles des solutions adoptées, qui sont dépendants des langages et des environnements utilisés.

Site web : http://www.plat-forms.org/

Le premier hackathon Perl en Europe, du 2 au 4 mars 2007

Jonathan Worthington a fait le chemin depuis l'Angleterre pour aller participer au premier *hackathon* Perl à Chicago, du 10 au 12 novembre 2006. Présent aux Journées Perl 2006 (les 25 et 26 novembre 2006), il a lancé l'idée d'un hackathon Perl en Europe. Sur les conseils de Mongueurs membres de la Foundation YAPC Europe, il a proposé l'idée à celle-ci. Ann Barcomb, l'une des organisatrices de YAPC Europe 2001 à Amsterdam, a pris la balle au bond, et organise donc le premier hackathon Perl européen.

Celui-ci se déroulera à Arnhem, en Hollande, du 2 au 4 mars 2007. On attend une vingtaine de participants pour faire avancer le développement de Perl 5, Perl 6, Parrot, et d'autres projets Perl.

Site web : http://conferences.yapceurope.org/hack2007nl/

Act (A Conference Toolkit) devient Open Source

Créé en 2004 par deux Mongueurs pour la première édition des Journées Perl, le logiciel Act (*A Conference Toolkit*) a été écrit pour éviter aux organisateurs de conférences de perdre leur temps à *hacker* le site web et leur permettre de se concentrer sur l'organisation de leur événement. S'il était écrit dans l'esprit *Open Source* depuis le début, seule l'envie de conserver une base de données centrale pour toutes les conférences Perl a freiné la diffusion du code.

Après plus de trois ans de développement et plus d'une douzaine de conférences hébergées, il était grand temps de diffuser le code, ce qui est fait depuis le 26 janvier dernier.

Site web : http://act.mongueurs.net/

Les prochaines conférences hébergées sur la plate-forme Act des Mongueurs de Perl ne seront pas moins que YAPC Europe, YAPC North America, le Nordic Perl Workshop et le premier hackathon Perl européen dont nous venons de parler. D'ailleurs, Act pourrait bien faire partie des projets sur lesquels travailleront les hackers réunis à Arnhem. :-)

Philippe Bruhat et Sébastien Aperghis-Tramoni,

book@mongueurs.net
sebastien@aperghis.net

Lionel Tricon

FUSE, développez vos systèmes de fichiers dans l'espace utilisateur

Comment décrire FUSE succinctement ? Hum ! On peut déjà dire ce qu'il n'est pas : FUSE n'est pas un émulateur du micro-ordinateur Spectrum. De la même façon, FUSE n'est pas un satellite d'observation de la NASA... ni même le titre d'un album rock de l'année 69 ou encore celui d'un film de nationalité bosniaque. Enfin, oui, c'est aussi tout cela (c'est fou comme les mots de 4 lettres peuvent être tellement de choses), mais en ce qui nous concerne, FUSE est surtout et avant tout une extension au noyau Linux autorisant la création de systèmes de fichiers s'exécutant dans l'espace utilisateur.

PREAMBULE

Car, qui n'a jamais rêvé de pouvoir implémenter toutes les fonctionnalités d'un système de fichiers dans l'espace utilisateur ?

En effet, écrire un système de fichiers nécessite en général le développement d'un *driver* et donc de lier son code à la destinée de celui du noyau. Entre autres désavantages, cela signifie restreindre, pour le plus grand nombre, la diffusion de votre système de fichiers aux nouvelles variantes du noyau Linux ; ce qui n'est sans doute pas la meilleure solution pour viser un public le plus large possible.

Exercice de style en soi particulièrement formateur, l'écriture d'un driver, même celui d'un système de fichiers, n'est pas une chose aisée. Outre les connaissances spécifiques nécessaires à ce type de développement, cela occupe pas mal de temps et épuise rapidement un stock non négligeable de café noir[1].

Pour enfoncer le clou, accéder à l'espace réservé du noyau est un exercice périlleux ; au moindre comportement inattendu de votre driver, le risque n'est pas nul de mettre en péril toute la stabilité du système d'exploitation.

Pour les systèmes de fichiers classiques qui lisent et écrivent des données sur des supports physiques, cela est difficilement contournable ; pour optimiser les performances, il est indispensable de se trouver au cœur des débats.

Mais pour les systèmes de fichiers dits « virtuels », qui ne manipulent pas directement les données, mais servent souvent d'interface avec des données abstraites ou distantes, on se prend à rêver d'une solution qui nous permettrait de nous soustraire de la nécessité impérieuse de passer par le noyau[2].

Et c'est justement de cette sacro-sainte dépendance au noyau que FUSE **[1]** se propose de nous soulager ; FUSE étant l'acronyme de « *Filesystem in USErspace* », soit « Système de fichiers en espace utilisateur » pour les plus anglophobes d'entre vous.

FUSE a fait officiellement son apparition dans le noyau Linux à partir de la version 2.6.14 et se compose d'un module noyau que l'on accède par l'intermédiaire du fichier spécial `/dev/fuse` et d'une bibliothèque en espace utilisateur `libfuse`.

FUSE a été dérivé de la bibliothèque `AVFS` **[2]** (*A Virtual FileSystem*) qui visait le développement d'une interface permettant à n'importe quel programme de lire des données présentes sur des disquettes, dans des fichiers compressés `gzip`, `tar`, `zip`, `bzip2`, `ar` et `rar2` ou des protocoles distants comme `ftp`, `http`, `webdav`, `rsh/rcp` et `ssh/scp`... cela sans recompiler son application (en somme, `gnome-vfs` sous GNOME ou `kioslaves` sous KDE, mais accessible depuis n'importe quelle application).

> AVFS :
>
> Techniquement, le projet valorise avant exécution du programme cible la variable d'environnement `LD_PRELOAD` qui permet de surcharger sélectivement les fonctions des autres bibliothèques partagées et donc de rediriger les appels système d'entrées-sorties vers ceux d'AVFS.

Depuis, le projet s'est scindé en deux parties distinctes, pour notre plus grand bonheur, en libérant la partie la plus prometteuse, à savoir FUSE !

Sous Linux, il existe une version 2.4.x qui peut être compilée sur les noyaux 2.4 et une version 2.6.x qui peut être compilée sur les noyaux 2.6. La dernière version en date est la 2.6.1 (sortie le 1er décembre 2006).

Coté licence, le module noyau ainsi que les utilitaires associés sont placés sous licence GPL, alors que la bibliothèque `libfuse` (liée dynamiquement avec les applications) est sous licence LGPL (ce qui autorise tout naturellement l'utilisation de FUSE dans des applications non GPL).

Pour information, Linux n'est pas le seul système d'exploitation à disposer de FUSE (bien que ce soit la plate-forme native) : un portage est heureusement disponible pour FreeBSD 6.x et 7.x. **[3]** ainsi que pour Mac OS X **[4]** (à noter que pour ce dernier, c'est GOOGLE qui a réalisé le portage). Il va sans dire que puisque chaque implémentation respecte la même interface, tous les systèmes de fichiers virtuels reposant sur FUSE fonctionnent sur n'importe laquelle des trois plateformes (moyennant une compilation native des exécutables). Cela étend le nombre d'utilisateurs potentiels pour chacune de ces applications et permet d'ores et déjà

1. Quoique, personnellement, je préfère le thé à la cerise...

2. A ranger dans cette catégorie, le protocole FTP ou le montage en local d'une image ISO, par exemple.

de profiter d'un nombre respectable de systèmes de fichiers virtuels respectant les interfaces FUSE.

Car, puisque cela fonctionne dans l'espace utilisateur, tout peut être transformé en système de fichiers. Cela va de l'accès en lecture/écriture au système de fichiers NTFS, à l'accès distant par SSH ou encore à des systèmes de fichiers plus improbables, comme la translation temps-réel de fichiers au format FLAC ou OGG vers le format MP3 (Yacufs), l'accès transparent à une base de données MySQL (MySQLFS) ou encore l'accès aux articles Wikipédia comme de simples fichiers (WikipediaFS) **[5]**...

Tout peut être transposable vers FUSE ; cela ouvre des horizons nouveaux uniquement limités par l'imagination des développeurs (et beaucoup s'accordent à dire que l'on en est qu'au début).

La facilité de développement d'un système de fichiers FUSE est étonnante de simplicité du moment que l'on en a intégré les principes. Le développement en espace utilisateur permet un développement plus aisé et facilite grandement les tests ; et cela sans endommager l'intégrité du système d'exploitation ou altérer ses données.

L'un des seuls défauts de cette solution est que les performances sont forcément moindres tout en restant raisonnables par rapport à une solution purement noyau ; en réalité, la différence est peu perceptible, vu de l'utilisateur, si l'on considère le type de systèmes de fichiers virtuels qui est visé par cette solution.

Plus généralement, FUSE est à mettre au même plan que le chargement dynamique des modules dans le noyau Linux. C'est une solution qui permet d'étendre à chaud les fonctionnalités de ce dernier (cf. rajouter de nouveaux systèmes de fichiers) et représente en cela une réelle évolution plus qu'une révolution.

Certains rappelleront à juste titre que le *Hurd* fait cela bien mieux par l'intermédiaire des *translators*, et d'autres que FUSE est un *hack* assez malin autour du noyau, mais que cela reste au final un hack. A fonctionnalités égales, il faut reconnaître que les deux solutions se valent, mais peut-être pas avec la même élégance.

Pour suspendre temporairement ce débat, on me souffle à mon oreillette qu'une version Hurd serait en cours de développement (en version vraiment préliminaire)... de même qu'un portage serait en cours pour OpenSolaris **[6]**.

Quoi qu'il en soit, il faut retenir que FUSE nous évite de tout développer dans le noyau et permet que notre code soit plus facile à implémenter, plus facile à déployer et plus facile à tester.

INSTALLATION DE FUSE

Première étape indispensable avant d'essayer un système de fichiers FUSE : vérifier que le module noyau est présent sur votre distribution. Bien que celui-ci ne soit disponible en standard que depuis le 2.6.14, certaines distributions l'ont déjà intégré dans des versions antérieures. C'est le cas de la SUSE 10.0 qui fournit le module dans sa version maison du noyau 2.6.13.

```
# /sbin/modprobe -l fuse
/lib/modules/2.6.13-15.12-default/kernel/fs/fuse/fuse.ko
```

Sinon, il vous reste à compiler le module et à l'installer sur votre système. L'opération n'est pas compliquée et fait appel au désormais récurent « `./configure && make && make install` » (cela va aussi installer de surcroît la bibliothèque et un utilitaire système).

Si le module est présent dans votre noyau Linux, la seconde étape va consister à installer les paquets contenant la bibliothèque `libfuse` et l'utilitaire `fusermount` nécessaires pour démonter les points de montage FUSE (étape inutile en cas d'installation à partir des sources).

Sur la SUSE, Fedora ou Mandriva, il vous faudra installer les paquets `fuse` et `fuse-devel` ; sur une Debian ou ses dérivées (Ubuntu, Kubuntu...), il vous faudra installer les paquets `libfuse2`, `libfuse-dev` et `fuse-utils`.

Si vous avez installé les sources, il existe un moyen assez simple d'interroger la version de l'`API` de la bibliothèque `libfuse` (la dernière en date étant la 2.6 ; on peut suivre l'historique sur **[7]**) :

```
# more /usr/include/fuse/fuse_common.h
/** Major version of FUSE library interface */
#define FUSE_MAJOR_VERSION 2
/** Minor version of FUSE library interface */
#define FUSE_MINOR_VERSION 5
```

La troisième et dernière étape consiste à rendre le fichier spécial `/dev/fuse` disponible pour toutes vos applications en lecture et écriture.

Pour charger directement le driver, un simple `modprobe` fera l'affaire (sous `root`) :

```
# modprobe fuse
# lsmod  | grep fuse
fuse                   33420  0
```

Pour connaître la version de l'API du module, vous pouvez consulter le tampon des messages du noyau :

```
# dmesg | grep fuse
fuse init (API version 7.6)
```

Pour rendre le chargement du driver persistant, ajoutez le nom du module dans le fichier `/etc/modules` sur Debian et ses dérivés :

```
$ echo fuse >> /etc/modules
```

Sur la SUSE, il sera nécessaire de rajouter le nom du module dans la variable `MODULES_LOADED_ON_BOOT` du fichier `/etc/sysconfig/kernel` pour forcer le chargement du driver à l'amorçage du système :

```
# grep MODULES_LOADED_ON_BOOT /etc/sysconfig/kernel
MODULES_LOADED_ON_BOOT="fuse"
```

Lorsque le module `fuse` est chargé, il n'est pas pour autant accessible par tous les utilisateurs pour une question de droits (sur la Debian, il faut que l'utilisateur appartienne au groupe `fuse`) ce qui est un comble pour une fonctionnalité qui est censée être accessible par tous les utilisateurs (car, comme on le verra plus loin, l'aspect sécurité est pris en compte par FUSE).

On va corriger cela en modifiant les droits du fichier spécial en jouant sur les paramètres UDEV :

```
$ cat /etc/udev/rules.d/40-fuse.rules
KERNEL=="fuse", NAME="%k", MODE="0666"
```

Il sera peut-être même nécessaire de changer les droits de l'utilitaire Fusermount :

```
$ chmod 4755 /usr/bin/fusermount
```

MISE EN PRATIQUE

Vous êtes fin prêt pour les tests pratiques.

Nous allons monter une image `iso` en lecture seule en s'appuyant sur l'application `fuseiso` que vous pourrez télécharger sur **[8]**.

En effet, à moins que celle-ci ne soit disponible sous forme binaire pour votre distribution, il est nécessaire de l'installer à partir des sources :

```
$ tar xjvf fuseiso-20061017.tar.bz2
$ cd fuseiso-20061017/
$ ./configure
$ make
$ make install
```

Si vous ne disposez pas d'une image `iso`, vous pouvez en créer une par la commande :

```
$ mkisofs -J -r -o /tmp/test.iso fuseiso-20061017/
```

Il reste alors à monter l'image `iso` sur un répertoire vide (de préférence), comme vous le feriez avec la commande `mount` :

```
$ mkdir -p /tmp/mount
$ fuseiso -n /tmp/test.iso /tmp/mount/
$ mount | grep fuse
fuseiso on /tmp/mount type fuse (rw,nosuid,nodev)
$ ls /tmp/mount
...
```

Démontez le système de fichier fait appel à la commande `fusermount` (la commande `umount` ne fonctionnera pas) :

```
$ fusermount  -u /tmp/mount/
```

Voilà, ce n'est pas plus compliqué que ça. N'importe quel utilisateur peut à sa convenance monter et démonter une image `iso` !

Il est même possible de monter automatiquement un tel système de fichiers dans `/etc/fstab` en utilisant le type de système de fichiers FUSE (par la magie du script `/sbin/mount.fuse`) :

```
fuseiso#/tmp/test.iso   /tmp/mount fuse user 0 0
```

ET LA SECURITE DANS TOUT CA ?

Plusieurs mécanismes sont mis en œuvre pour améliorer la sécurité et faire que FUSE ne devienne trop perméable aux attaques :

* Par défaut, seul l'utilisateur ayant effectué un montage peux accéder ou démonter ce point de montage (même le super-utilisateur en est exclu !). Cela sécurise l'accès aux données.

* Les options `nosuid` et `nodev` sont forcées par défaut lors du montage. Il n'est pas possible de positionner un bit `suid` pour prendre l'identité de quelqu'un d'autre de même que les fichiers spéciaux ne sont pas interprétés sur ce type de système de fichiers.

Pour autoriser tous les utilisateurs à accéder au point de montage, il faut ajouter l'option `allow_other` lors de l'appel à notre application :

```
fuseiso -n /tmp/test.iso /tmp/mount/ -o allow_other
```

Par symétrie, autoriser l'accès en super-utilisateur nécessite l'utilisation de l'option `allow_root` sachant que ces deux paramètres ne sont pris en compte que lorsque le fichier `/etc/fuse.conf` est valorisé avec l'option `user_allow_other`.

Le fait que l'utilitaire `fusermount` possède un `suid-bit` positionné à *root* peut difficilement être contourné, mais, finalement, comme c'est exactement la même chose pour l'utilitaire `mount`, ce n'est pas en soi un véritable problème, car lié à la conception même du noyau Linux.

A PROPOS, COMMENT CA MARCHE ?

L'API native est écrite en C avec de nombreux *binding* vers le C++, le Java, le C#, ainsi que des langages scripts tels que Python, Perl, TCL ou Ruby (cette liste n'est pas exhaustive).

Prenons un exemple pédagogique. Imaginez que nous souhaitions implémenter un module FUSE qui présente sous forme de fichiers tous les utilisateurs du système avec, comme contenu, la description des utilisateurs (dans cette optique, il nous vient tout naturellement à l'esprit d'utiliser la base de données que constitue le fichier `/etc/passwd`).

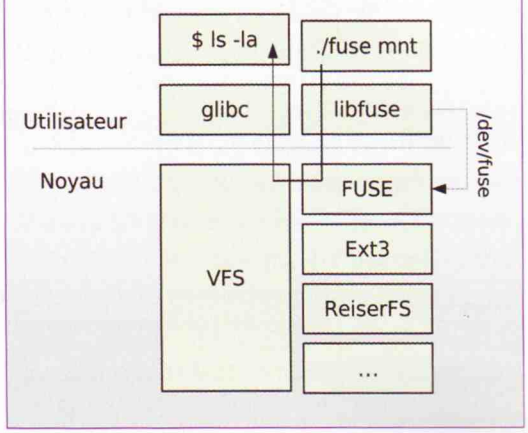
Pour cela, notre programme, écrit en langage C et réduit à sa plus simple expression pour faciliter la lecture, va implémenter les appels système `readdir`, `open` et `read`. Il faut rajouter à cela l'appel `getattr` qui n'existe pas dans la `libc` et qui doit être vu comme l'équivalent de l'appel système `stat`. Cette fonction occupe une place toute particulière dans FUSE, car elle sera appelée avant tout accès à un chemin dans le système de fichiers (il est alors possible de centraliser pas mal de traitements dans cette fonction suivant le type de système de fichiers que l'on implémente).

Toutes les fonctions sont alors regroupées dans une structure de type `fuse_operations`, puis passées en paramètre à la fonction bloquante `fuse_main()`, laquelle va gérer tous les évènements sur le système de fichiers en collaboration avec le driver `fuse` (figure 1).

Comme précisé dans la figure 1, ces appels système vont remplacer ceux de la bibliothèque C standard par l'intermédiaire de la bibliothèque `libfuse` (elle-même interfacée avec le fichier spécial `/dev/fuse`).

Voilà comment il est possible d'implémenter un tel système de fichiers sous FUSE :

Figure1 : Interaction avec le driver fuse et la libc




```
#include <fuse.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
#include <pwd.h>

struct passwd *pw;

static int passwd_getattr(const char *path, struct stat *stbuf)
{
    memset(stbuf, 0, sizeof(struct stat));
    if(strcmp(path, "/") == 0) {
        stbuf->st_mode = S_IFDIR | 0755;
        stbuf->st_nlink = 2;
          return 0;
    }
    pw = getpwnam( &(path[1]) );
    if (pw == NULL) return -ENOENT;

    stbuf->st_mode = S_IFREG | 0444;
    stbuf->st_nlink = 1;
    stbuf->st_uid = getuid();
    stbuf->st_gid = getgid();
    stbuf->st_size = (off_t)strlen(pw->pw_gecos)+1;
    return 0;
}

static int passwd_readdir(const char *path, void *buf,
fuse_fill_dir_t filler, off_t offset, struct fuse_file_info *fi)
{
    (void) offset;
    (void) fi;

    if(strcmp(path, "/") != 0) return -ENOENT;

    filler(buf, ".", NULL, 0);
    filler(buf, "..", NULL, 0);
    setpwent();
    while ((pw = getpwent())) filler(buf,pw->pw_name,NULL,0);
    endpwent();

    return 0;
}


static int passwd_open(const char *path, struct fuse_file_info
*fi)
{
    pw = getpwnam( &(path[1]) );
    if (pw == NULL) return -ENOENT;
    if ((fi->flags & 3) != O_RDONLY)
        return -EACCES;
    return 0;
}

static int passwd_read(const char *path, char *buf, size_t size,
off_t offset, struct fuse_file_info *fi)
{
    size_t len;
    (void) fi;

    pw = getpwnam( &(path[1]) );
    if (pw == NULL) return -ENOENT;

    len = (size_t)strlen(pw->pw_gecos)+1;
    if (offset < len) {
        if (offset+size > len) size=len-offset;
        memcpy( buf, (pw->pw_gecos)+offset, size );
          buf[size-1] = '\n';
        return size;
    }
    return 0;
}

static struct fuse_operations passwd_oper = {
    .getattr         = passwd_getattr,
    .readdir         = passwd_readdir,
    .open  = passwd_open,
    .read  = passwd_read,
};

int main(int argc, char *argv[])
{
    return fuse_main(argc, argv, &passwd_oper);
}
```

Hormis la spécificité de l'appel à la fonction `fuse_main()` avec le passage en argument de la liste des appels système qui seront utilisés par le système de fichiers (structure `fuse_operations`), rien n'est fondamentalement compliqué à appréhender dans ce programme pour quelqu'un qui manipule raisonnablement le langage C.

La liste et les prototypes des appels système qu'il est possible d'implémenter sont disponibles dans le fichier `/usr/include/fuse/fuse.h`. Je vous invite cordialement à y faire un petit tour.

Les puristes n'auront pas oublié de noter que, pour bien faire, il faudrait utiliser les versions ré-entrantes des primitives qui manipulent le fichier `/etc/passwd`. Cela a volontairement été omis dans le listing précédent par souci de clarté.

Pour générer l'exécutable, tapez la commande suivante dans une console :

```
$ gcc -Wall -o fuse fuse.c -lfuse
```

L'utilisation de l'option `-D_FILE_OFFSET_BITS=64` sera sans doute nécessaire pour compiler correctement FUSE. Cette option, utile au compilateur, permet de demander de remplacer de façon transparente les appels système 32 bits vers ceux 64 bits, et ce, même sur architecture 32 bits (par exemple, l'appel système `stat64` sera renommé en `stat`).

```
/usr/include/fuse/fuse_common.h:30:2: error: #error Please add
-D_FILE_OFFSET_BITS=64 to your compile flags!
```

Si le programme ne veut toujours pas compiler, l'astuce consiste alors à préciser au compilateur quelle version de l'interface de programmation `libfuse` on souhaite utiliser en valorisant l'option `-DFUSE_USE_VERSION=XX`. Sur la machine de test, les versions 22, 25 et 26 ont donné de bon résultats.

Passons à la mise en pratique de notre programme :

```
$ mkdir mnt
$ ./fuse mnt/
$ cd mnt/
$ ls
at      bin         games   laurence   lionel    mail  wwwrun
man     messagebus  news    ntp        postfix   root  uucp
avahi   daemon      ftp     haldaemon  ldap      lp    vdr
mdnsd   mysql       nobody  nx         postgres  sshd
$ ls -la games
-r--r--r--  1 lionel users 14 1970-01-01 01:00 games
$ cat games
Games account
$ cd ../
$ fusermount -u mnt
```

A noter aussi que le programme (dont les sources sont disponibles sur le site **[9]**) peut être appelé avec de nombreuses options que l'on peut consulter avec le flag `-h`. En particulier, l'option `-d` permet d'obtenir pas mal d'informations intéressantes sur le fonctionnement interne du programme qui sinon passe immédiatement en tâche de fond.

ET VOUS ? VOUS PRENDREZ QUOI COMME SYSTEMES DE FICHIERS ?

Référencés sur le site officiel, il existe actuellement un peu moins d'une centaine de systèmes de fichiers conçus au-dessus de FUSE. Dans la réalité, ils doivent être un peu plus nombreux.

Tous ne présentent peut-être pas le même intérêt pratique, et certains sont même d'inspirations douteuses, mais cela signifie surtout que cette technologie peu connue ouvre une fenêtre vers une multitude d'expérimentations en tout genre qui devrait nous offrir quelques belles pépites et étendre un peu plus la puissance et la modularité de notre OS favori.

Un petit passage en revue d'un florilège de système de fichiers **[10]** qui devraient vous donner envie de vous intéresser à cette technologie :

NOM	DESCRIPTION
SSHFS	Parcourir une arborescence distante par l'intermédiaire du protocole SFTP
BlogFS	Pour consulter les *blogs* WordPress comme de simples fichiers
Captive NTFS	Lire et écrire sur des partitions NTFS au travers du driver natif Windows
Cddfs	Monter vos CD-Rom audio pour accéder au contenu WAV de façon transparente
EncFS	Système de fichier crypté pour sécuriser vos données
FuseDAV	Parcourir les partages WebDAV
Fusecram	Monter les images compressées au format cramfs
FuseCompress	Compresser des données à la volée
GnomeVFS2	Monter tous les systèmes de fichiers accessibles par Nautilus sous GNOME
gphoto2-fuse-fs	Permet de d'accéder à un appareil photo numérique supporté par gPhoto2 sans passer par gtkam ou gPhoto2
Kio Fuse Gateway	Monter les *ioslaves* KDE pour les exposer aux autres applications
Mountlo	Monter des fichiers en *loopback*
WikipediaFS	Voir et éditer les articles Wikipedia comme de simples fichiers
Yacufs	Convertit à la volée tous vos fichiers `.ogg`, `.flac` en fichiers MP3

Un dernier mot avant de refermer cet article ? Le mot FUSE signifie « fusible » en anglais et c'est peut-être ce qui le caractérise le mieux ; la capacité de développer et d'ajouter de nouveaux systèmes de fichiers sous Linux sans mise à jour du noyau et, surtout, sans mettre en péril sa robustesse. À vos claviers !

Lionel Tricon,

lionel.tricon@free.fr

LIENS

- [1] http://fuse.sourceforge.net/
- [2] http://sourceforge.net/projects/avfs
- [3] http://fuse4bsd.creo.hu/
- [4] http://code.google.com/p/macfuse/
- [5] http://wikipediafs.sourceforge.net/
- [6] http://www.opensolaris.org/os/project/fuse/
- [7] http://fuse.sourceforge.net/wiki/index.php/ApiChangelog
- [8] http://freshmeat.net/projects/fuseiso/
- [9] http://lionel.tricon.free.fr/Articles/fuse/exemple
- [10] http://fuse.sourceforge.net/wiki/index.php/FileSystems

Olivier Saraja

Yafray, le moteur de rendu photoréaliste libre : maîtriser les shaders et les propriétés matériau

Depuis le début de cette série, nous présentons Yafray comme un moteur de rendu photoréaliste. Nous avons étudié sa capacité à reproduire des schémas d'éclairage crédibles, à simuler le flou focal d'une véritable caméra, mais sans pour autant s'attacher à l'essentiel : les shaders et les propriétés matériau.

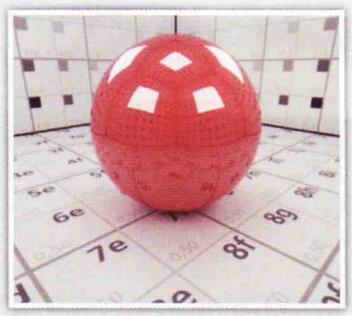
En effet, c'est, en premier, le soin apporté à la création des *shaders* qui va déterminer l'apparence de vos surfaces dans vos scènes et donc suggérer une notion de réalisme à l'observateur de l'image. Mais qu'est-ce qu'un shader ? Il s'agit tout simplement d'un terme générique des propriétés de la surface d'un objet qui indique au moteur de rendu comment ombrer celle-ci (ombre se traduit par *shadow*, en anglais) en fonction des sources d'éclairage. Le shader prend en compte les propriétés de diffusion de la lumière du matériau, ainsi que sa restitution à la surface de l'objet : certaines ombres peuvent être plus diffuses ou, au contraire, plus tranchées. Le shader prend également en compte les reflets spéculaires, c'est-à-dire les taches lumineuses à la surface des objets éclairés. Côté matériau, le moteur de rendu prend également en compte les propriétés de réflexion (miroir) ou de réfraction (verre) de la lumière, grâce au *raytracing*, sachant que ces deux propriétés sont les fondations d'un rendu photoréaliste, car capables de restituer fidèlement les mêmes phénomènes dans notre réalité. Enfin, d'autres propriétés de matériau sont également simulées par Yafray, comme par exemple l'effet Fresnel pour les matériaux réfléchissants, ou l'absorption, le filtrage ou la diffraction lumineuse pour les matériaux transparents. La dispersion subsurfacique (*subsurface scattering* ou SSS) est également supportée de façon expérimentale.

ASTUCE

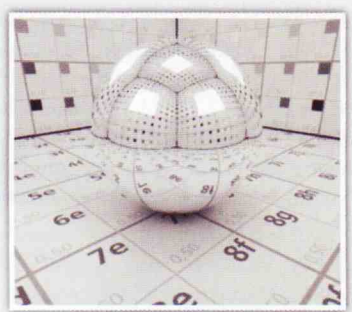
Matériaux prédéfinis dans Blender

Si vous utilisez Yafray dans le cadre de la suite de modélisation, animation et rendu Blender, vous trouverez dans l'onglet *Mirror Transp* (*Material buttons*, menu *Shading [F5]*) un sélecteur de matériaux prédéterminés. Parmi ceux-ci, on retrouve les prédéfinitions qui suivent. Vous noterez qu'une couleur de base rouge lui a été attribuée, pour une meilleure lisibilité des images tests, et qu'il est intéressant de noter la couleur des caustiques réfractés et/ou réfléchis, de même que l'éclairage indirect obtenu grâce à l'illumination globale.

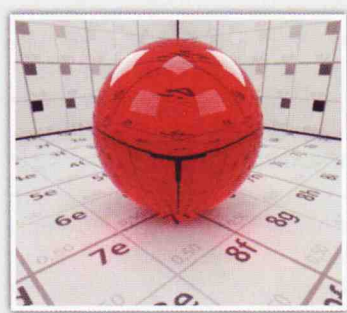
Fresnel reflect : convenant aux objets colorés et réfléchissants (or, bronze, autres métaux)



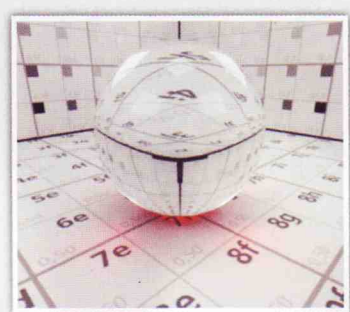
Uniform reflect : convenant aux objets parfaitement réfléchissants, sans couleur propre (miroir, chrome, etc.)



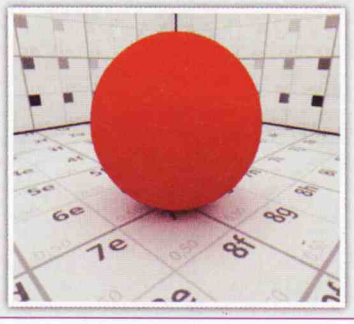
Color Glass : convenant aux objets colorés et transparents (bouteille de vin, etc.)



Clear Glass : convenant aux objets parfaitement transparents, sans couleur propre (vitres, verres, etc.)



No Reflect/transmit : matériau vierge servant à la base de la création de tous autres types de surfaces



1. Côté shaders

Beaucoup de débutants pensent à tort que le fait d'utiliser un moteur de rendu photoréaliste va rendre leurs œuvres, sans effort particulier, également photoréalistes. C'est bien évidemment une erreur, dans le sens où nul logiciel de 3D ne propose dans son interface un bouton magique portant la mention *Faire une photo*. Le photoréalisme se travaille et se gagne sur les détails, que ce soit au niveau de la modélisation, des textures, de la mise en scène ou, encore, des matériaux. Et comme souvent en matière de 3D, c'est l'observation de la réalité qui nous entoure qui permet à l'artiste de choisir les effets à obtenir pour qu'un observateur admette comme réaliste une scène quelconque.

Cela est d'autant plus vrai au niveau du choix et des réglages des shaders. De nombreux algorithmes d'ombrage des surfaces existent, et certains bénéficient plus de la faveur du grand public que d'autres, en fonction de leur simplicité de mise en œuvre et du résultat obtenu. Mais chacun a été développé pour favoriser une méthode de diffusion de la lumière à la surface des objets, propre à tel ou tel matériau réel. C'est donc là que le sens de l'observation de l'artiste se révèle incontournable : il lui faudra choisir tel shader plutôt qu'un autre, en fonction de la nature des objets qu'il souhaite mettre en scène.

Étudions un bloc `<shader></shader>` typique de Yafray, par exemple le bloc suivant, qui contient tous les attributs les plus courants (ou ceux par défaut) à partir desquels il est possible de définir des matériaux simples :

```
<shader type="blendershader" name="MAMaterial" >
    <attributes>
        <color r="0.800000" g="0.800000" b="0.800000" />
        <specular_color r="1.000000" g="1.000000" b="1.000000" />
        <mirror_color r="1.000000" g="1.000000" b="1.000000" />
                <diffuse_reflect value="0.800000" />
                <specular_amount value="0.500000" />
                <alpha value="1.000000" />
                <emit value="0.000000" />
                <matmodes value="traceable shadow" />
                <diffuse_brdf value="lambert" />
                <specular_brdf value="blender_cooktorr" />
                <hard value="50" />
        </attributes>
</shader>
```

Les couleurs de l'objet sont déterminées à partir de trois attributs, chacun doté des trois composantes de couleur de base : `r` pour *red* (rouge), `g` pour *green* (vert) et enfin `b` pour *blue* (bleu).

- `color` : définit la couleur de base de l'objet, c'est-à-dire celle qui est visible lorsque tous ses pixels sont 100% éclairés, sans ombre ni spéculaires ou rehaut lumineux ;
- `specular_color` : définit la couleur de la tache spéculaire, ce qui est utile pour simuler des objets métalliques ou certains matériaux organiques ;
- `mirror_color` : définit la couleur qui teinte les reflets à la surface de l'objet.

Suivent ensuite quelques propriétés typiques, communes à tous les shaders :

- `diffuse_reflect` : définit le pourcentage de lumière réfléchie par le matériau ; une faible valeur implique que peu de lumière est réfléchie : l'objet apparaît sombre. Une valeur élevée implique que beaucoup de lumière est réfléchie : l'objet apparaît plus clairement.
- `specular_amount`: définit la spécularité de l'objet ; plus cette valeur sera élevée, plus les taches spéculaires seront intenses et visibles à la surface de l'objet, lorsque illuminée par une lampe.
- `alpha` : définit la transparence de l'objet. Une valeur nulle indique un objet totalement invisible, une valeur de 1.000, quant à elle, indique un objet totalement opaque.
- `emit` : définit l'émittance de l'objet, c'est-à-dire la portion de luminosité propre (en dehors de tout système d'éclairage) qu'il émet. Utile pour l'illumination globale lorsque aucune lampe n'est placée dans la scène et que l'on souhaite que ce soit un maillage qui soit émetteur de lumière (écran cathodique, lampe de chevet, ou autre). Attention : des valeurs très faibles (de l'ordre de 0.100) suffisent déjà à éclairer des scènes entières (voir *GNU/Linux Magazine* n°90 pour l'article expliquant le fonctionnement de l'illumination globale).

1.1 Shader Diffus

Le shader diffus est celui qui va déterminer la forme et la vitesse de propagation de la lumière à la surface d'un objet éclairé. A noter que le terme BRDF (*Bidirectional Reflectance Distribution Function*) se substitue parfois au terme shader. Yafray accepte presque tous les shaders diffus de Blender, c'est-à-dire, plus particulièrement :

- `Minnaert` : ce type de shader convient particulièrement à la simulation de surfaces poreuses ou des surfaces poussiéreuses, des métaux rouillés, du bois brut non verni ou encore du ciment. Il admet notamment un paramètre supplémentaire `darkening`.
- `Toon` : ce type de shader ne se raccroche à rien de physiquement observable dans la réalité. Il s'agit d'une approche artistique des matériaux.

- `Oren-Nayar` : ce type de shader convient parfaitement aux tissus, au bois, à la peau ou au velours. Il admet notamment un paramètre supplémentaire `roughness`.
- `Lambert` : ce type de shader est parfait pour les plastiques, les matériaux artificiels et le marbre poli. On l'appelle également le diffuseur parfait.

Le type de shader diffus est déterminé par un attribut qui prend la forme suivante :

```
<diffuse_brdf value="[shader]" />
```

où `[shader]` peut être remplacé par `lambert`, `minnaert`, `toon` et `oren_nayar`. Les types de shader présents dans Blender, mais non supportés par Yafray sont remplacés par le shader par défaut `lambert`.

1.2 Shader Spéculaire

Le shader spéculaire va définir le modèle de spéculaire utilisé. Yafray accepte presque tous les shaders spéculaires de Blender, c'est-à-dire :

- `Wardiso` : ce type de shader convient surtout aux matériaux anisotropes, avec des rehauts mieux définis. Il admet deux paramètres supplémentaires : `u_roughness` et `v_roughness` qui spécifient la rugosité du matériau dans les deux directions principales de la surface. Par exemple :

```
<specular_brdf value="ward" />
<u_roughness value="0.100000" />
<v_roughness value="0.100000" />
```

- `Toon` : ce type de shader ne se raccroche à rien de physiquement observable dans la réalité. Il s'agit d'une approche artistique des matériaux.
- `Phong` : Ce type de shader est particulièrement adapté aux peintures brillantes, aux laques et aux vernis, aux plastiques, au métal poli et à certains polymères. Il admet un paramètre supplémentaire, `hard`, pour déterminer la dureté de la tache spéculaire. Par exemple :

```
<specular_brdf value="phong" />
<hard value="50" />
```

- `Cook-torrance` : ce type de shader est plus particulièrement adapté aux matériaux avec une très forte spécularité, comme les carrosseries de voiture, le verre, l'eau ; les matériaux avec une moindre spécularité, comme le métal non poli ou le verre dépoli ; et enfin aux matériaux dont les reflets sont plus diffus, comme le bois, la pierre ou la peau. Il admet un paramètre supplémentaire, `hard`, pour déterminer la dureté de la tache spéculaire. Par exemple :

```
<specular_brdf value="blender_cooktorr" />
<hard value="50" />
```

- `Blinn` : ce type de shader repose sur un modèle plus physique que les autres, et admet notamment un paramètre supplémentaire, `blinn_ior`, pour déterminer l'indice de réfraction du matériau pour la détermination du reflet spéculaire, et `hard` pour déterminer la dureté de la tache spéculaire. Par exemple :

```
<specular_brdf value="blinn" />
<blinn_ior value="4.000000" />
<hard value="50" />
```

Le type de shader spéculaire est déterminé par un attribut qui prend la forme suivante :

```
<diffuse_brdf value="[shader]" />
```

où `[shader]` peut être remplacé par `blender_cooktorr`, `phong`, `blinn`, `toon` et `ward`.

1.3 Le subsurface scattering

La dispersion subsurfacique (*subsurface scattering* ou SSS) est un phénomène apparaissant lorsque de la lumière traverse la surface externe d'un objet et propage plus ou moins fortement au travers de la matière même. Elle peut être observée sur des matériaux parfaitement monolithiques (certains minéraux, comme le jade, par exemple) ou à la surface d'objets composés de plusieurs matériaux (comme la peau humaine, dont la couche supérieure est translucide et laisse passer une quantité certaine de lumière), mais aussi les végétaux.

La mise en place d'un tel effet avec Yafray demande un tout petit peu plus de travail que les autres shaders. En effet, l'effet est piloté dans un shader à part, auquel il sera fait appel par le shader de l'objet affecté. Ainsi, copiez ce bloc `<shader>` dans votre fichier `.xml`, en amont du bloc `<shader>` de l'objet affecté :

```
<shader type="sss" name="exemple_sss">
          <attributes>
          <color r="1.0" g="0.0" b="0.0"/>
          <radius value="1.5"/>
          <samples value="100"/>
          </attributes>
</shader>
```

Ensuite, dans le bloc `<shader>` de l'objet affecté, vous ajouterez une simple ligne :

```
<environment value="exemple_sss" />
```

Par exemple :

```
<shader type="sss" name="exemple_sss">
          <attributes>
          <color r="1.0" g="0.0" b="0.0"/>
          <radius value="1.5"/>
          <samples value="100"/>
          </attributes>
</shader>
<shader type="blendershader" name="MAMaterial.001" >
    <attributes>
          <color r="0.936526" g="0.720222" b="0.676962" />
          <specular_color r="1.000000" g="1.000000" b="1.000000" />
          <mirror_color r="1.000000" g="1.000000" b="1.000000" />
          <diffuse_reflect value="0.800000" />
          <specular_amount value="0.150000" />
          <alpha value="1.000000" />
          <emit value="0.000000" />
          <matmodes value="traceable shadow" />
          <diffuse_brdf value="oren_nayar" />
          <roughness value="0.100000" />
          <specular_brdf value="blender_cooktorr" />
```

```
            <hard value="50" />
            <environment value="exemple_sss" />
        </attributes>
</shader>
```

Figure 01 : Exemple de matériau utilisant le subsurface scattering

Les paramètres sont assez simples : `color` permet de déterminer la couleur diffusée sous la surface (par exemple, vous choisirez un rouge bien prononcé pour de la peau humaine) ; `radius` permet de définir la densité/translucidité apparente de l'objet ; enfin, `samples` permet de spécifier le nombre d'échantillonnages à effectuer. Attention : plus la valeur de radius sera élevée, plus vous aurez besoin d'échantillons pour obtenir des résultats convenables.

Vous noterez enfin que le shader `sss`, tel qu'implémenté à ce jour dans Yafray, n'est ni physiquement correct, ni totalement terminé d'implémentation. Selon l'auteur même, s'il donne des résultats satisfaisants, c'est plus dû à la chance qu'à un code complet et bien établi.

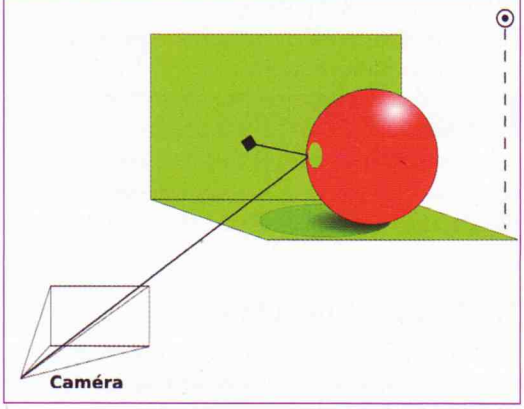
2. Côté matériau

Pendant de longues années, le raytracing est resté le roi indétrônable du rendu, car il était la seule méthode abordable pour obtenir de véritables reflets, des réfractions réalistes et des ombres parfaites ; ces caractéristiques représentaient alors le summum du réalisme. Mais avec l'évolution de la micro-informatique offrant toujours plus de puissance de calcul, permettant maintenant, notamment, de mettre en place des solutions d'illumination globale, de caustiques ou encore de flou focal, ces mêmes caractéristiques ne sont plus aujourd'hui qu'une composante, non négligeable, certes, de ce que l'on admet couramment être du photoréalisme.

2.1 La réflexion

Le tracé d'un reflet à la surface d'un objet répond à une méthodologie assez simple à comprendre, ainsi qu'à mettre en place grâce au principe du raytracing. Comme d'habitude, un rayon est lancé de la caméra en direction du pixel équivalent de l'image finale. En vulgarisant au maximum, si le rayon rencontre une surface, le moteur va inspecter ses propriétés matériau pour déterminer si la surface est réflective. Si oui, le rayon rebondira sur la surface (en fonction de l'angle incident entre le rayon et la normale à la surface) à la recherche d'une autre surface dans son environnement.

Figure 02 : Principe de la détermination de la couleur d'un reflet à la surface d'un objet réfléchissant



Bien sûr, si la surface trouvée par le rayon après son rebond est elle-même réfléchissante, le rayon rebondit une nouvelle fois, et ainsi de suite, jusqu'à trouver une surface qui n'est pas réfléchissante. Bien sûr, le pixel de l'image finale se voit attribuer la couleur de la surface (au point de contact) du dernier objet finalement trouvé, sauf si la surface réfléchissante est elle-même colorée, ou si l'objet n'est que partiellement réfléchissant (en ce cas, le reflet sera plus ou moins teinté par les couleurs propres des divers objets réfléchissants).

Enfin, dans un environnement constitué uniquement d'objets réfléchissants, un même rayon peut rebondir indéfiniment, et le rendu nécessiter un temps théoriquement infini pour être réalisé. Il est possible de limiter artificiellement le nombre d'itérations pour éviter que cela ne se produise.

2.1.1 Les reflets

La réflexion d'un objet est définie par un attribut `reflect_amount`. Une valeur nulle induit l'absence totale de reflets, tandis qu'une valeur égale à `1.000` suggère un miroir parfait.

```
<reflect value="on" />
<reflect_amount value="1.000000" />
```

Figure 03 : De l'objet totalement dépoli au miroir parfait, l'attribut reflect_amount permet de fixer le taux de réflectivité des surfaces (de gauche à droite, des valeurs respectivement égales à 0.000, 0.500, et 1.000).

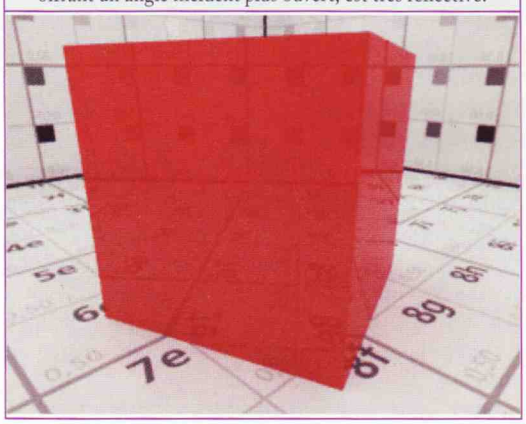
2.1.2 L'effet Fresnel

L'effet Fresnel est un phénomène qui rend le reflet observable seulement lorsque l'angle incident entre le vecteur de la caméra et la normale à l'objet au point

considéré atteint un certain angle. Cela permet de simuler certains matériaux (en particulier des objets vernis ou laqués) qui ne semblent réfléchissants que sous certains angles d'observation.

```
<fresnel_offset value="5.000000" />
```

Figure 04 : Illustration de l'effet Fresnel sur un cube : la face frontale, pratiquement normale à l'angle d'observation de la caméra, est presque mate. En revanche, la face latérale, offrant un angle incident plus ouvert, est très réflective.



2.1.3 Le Conetrace

Cette fonction assez peu connue permet d'obtenir des reflets granuleux ou légèrement troubles, sans pour autant avoir recours à une carte de bosselage (*bump map*), par exemple. Dans l'esprit, il fonctionne comme le shader `sss` : il faut dans un premier temps mettre en place un bloc `<shader>` pour le `conetrace` :

```
<shader type="conetrace" name="env" reflect="on"
angle="7" samples="5" IOR="1">
                    <attributes>
                    <color r="0.5" g="0.5" b="0.5" />
                    </attributes>
</shader>
```

en amont du shader de l'objet affecté. Celui-ci verra son propre shader complété par l'attribut suivant :

```
<environment value="env" />
```

Figure 05 : Exemple de reflets flous grâce au shader conetrace à gauche, et reflet normal à droite

Les attributs sont les suivants :

- `reflect` : la valeur `"on"` permet d'activer l'effet de granularité des reflets. `"off"` indique la réfraction des rayons.
- `angle` : indique le degré de distorsion ou de granularité ; une valeur nulle suggère une clarté totale des reflets.
- `samples` : indique le nombre d'échantillons appliqués au reflet.
- `IOR` : indique l'indice de réfraction des rayons.
- `color` : indique le filtrage de la lumière arrivant sur la surface.

2.2 La transparence

Le principe du rendu en raytracing d'une surface transparente est proche de celui d'une surface réfléchissante, dans l'esprit. De la même façon qu'auparavant un rayon sera lancé depuis la caméra en direction d'un pixel de l'image finale, sauf que lorsqu'il rencontrera un objet possédant une propriété de transparence, le rayon poursuivra sa course vers l'environnement en arrière-plan, sans rebondir sur sa surface.

A nouveau, si la surface trouvée par le rayon après avoir traversé le premier objet est également transparente, le rayon va poursuivre sa course jusqu'à rencontrer un objet totalement opaque. Bien sûr, le pixel de l'image finale se voit attribué la couleur de la surface (au point de contact) du premier objet opaque rencontré, sauf si la surface transparente est elle-même colorée, ou si l'objet n'est que partiellement transparent (en ce cas, la scène vue par transparence sera plus ou moins teintée par les couleurs propres des divers objets transparents placés les uns derrière les autres.

Tel qu'exposé, ce principe est toutefois incomplet. En effet, lorsque l'on regarde au travers d'un objet transparent massif dont les surfaces sont courbes, on n'observe pas, au point considéré, la scène strictement placée derrière l'objet, mais une autre partie de la scène. Ce phénomène très connu (et exploité notamment en optique), appelé « réfraction », est dû au fait que les rayons de lumière sont déviés par la masse transparente, en fonction de ses formes, mais aussi de sa densité. Chaque matériau possède en effet une densité pouvant affecter le trajet de la lumière dans son corps ; cette propriété est communément nommée indice de réfraction du matériau (ou IoR, *index of refraction*), et est, par exemple, égale à 1.000 pour de l'air pur, environ 1.500 pour du verre et environ 1.330 pour de l'eau pure.

Indices de réfraction

À titre indicatif, les indices de réfraction suivants sont utiles à connaître : Air 1.0002926, Alcool 1.329, Diamant 2.417, Émeraude 1.576, Verre 1.51714, Plastique 1.460, Plexiglas 1.50, Quartz 1.544, Eau 1.33335.

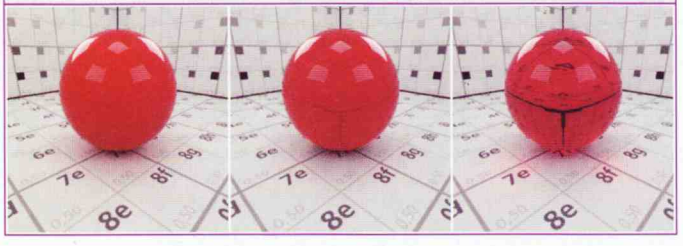
En raytracing, lorsque le rayon traverse un objet transparent, au lieu de continuer imperturbablement sa course initiale, il sera plus ou moins fortement dévié par la masse transparente pour finalement aller frapper la surface d'un objet qui ne sera pas forcément placée immédiatement derrière l'objet transparent. C'est le phénomène de réfraction.

2.2.1 La transparence

La transparence est déterminée par un attribut `alpha`. Une valeur nulle indique un objet totalement invisible, tandis qu'une valeur égale à `1.000` indique un objet totalement opaque. Pour des matériaux comme le verre, des valeurs très faibles seront requises :

```
<alpha value="0.001000" />
```

Figure 06 : Alpha respectivement égal à 1.000, 0.500 et 0.001.

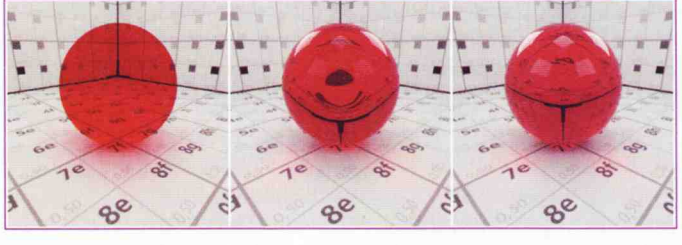


2.2.2 La réfraction

L'indice de réfraction détermine le taux de distorsion du trajet d'un rayon lumineux au travers d'un objet transparent. Une valeur égale à `1.000` indique l'absence totale de distorsion, tandis que des valeurs supérieures indiquent une distorsion croissante.

```
<refract value="on" />
<IOR value="1.500000" />
```

Figure 07 : IoR respectivement égal à 1.000 (air), 1.333 (eau) et 1.500 (verre) : on observe clairement la déformation de l'environnement au travers de l'objet transparent.



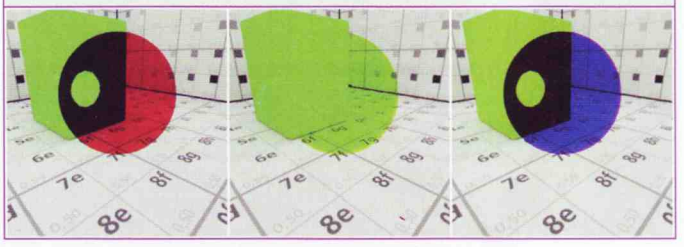
2.2.3 L'absorption et le filtrage

L'absorption est une propriété des matériaux transparents qui laissent passer une composante ou plusieurs (Rouge, Verte ou Bleue) des couleurs situées en arrière-plan, et bloquent totalement les autres, de sorte qu'elles ressortent noires. Par exemple, imaginons (figure 08, ci-après) une sphère transparente laissant passer le Rouge et absorbant le Vert et le Bleu. Un cube Vert (Pur ! Pas de Rouge, ni de Bleu parmi ses composantes) sera totalement bloqué (il apparaît noir), tandis que le sol blanc (couleur issue du mélange des composantes Vert, Bleu et Rouge) paraîtra rouge (ses composantes Vert et Bleu sont bloquées, mais sa composante Rouge est visible). Par exemple :

```
<absorption r="1.000000" g="1.000000" b="1.000000" />
```

laisse passer toutes les composantes de couleur sans les bloquer.

Figure 08 : Le paramètre Absorption permet de définir une couleur visible au travers de l'objet transparent, et de bloquer toutes les autres ; ne pas oublier, dans ces images, que le blanc du sol est obtenu par mélange des trois composantes fondamentales : Rouge, Vert et Bleu.

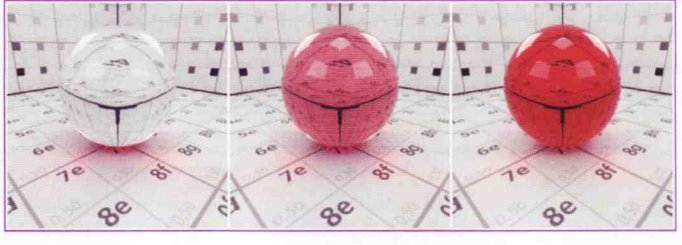


Un objet transparent laisse normalement percevoir l'environnement placé derrière lui sans en altérer les couleurs. Ce n'est toutefois pas un comportement réaliste, surtout lorsque l'on simule des objets en verre teintés dans la masse (comme par exemple une bouteille de vin vert, ou une bouteille de bière brune). C'est à cela que sert le paramètre `transmit_filter` : à déterminer quelle proportion de la couleur de base du matériau doit teinter l'environnement perçu au travers de l'objet. Par exemple :

```
<transmit_filter value="1.000000" />
```

teinte totalement, dans la masse, l'environnement perçu au travers de l'objet transparent coloré.

Figure 09 : Le transmit_filter permet de déterminer la proportion de la couleur de base du matériau qui teinte les objets vus au travers de l'objet ; de gauche à droite, des valeurs respectivement égales à 0.000, 0.500 et 1.000.



transmit_filter et les taches caustiques

Vous noterez que dans l'établissement de la carte des photons, c'est la couleur de base (rouge) du matériau qui est prise en compte, ainsi qu'en témoignent les taches caustiques sous la sphère. Avec une valeur `transmit_filter` nulle, toutefois, la sphère apparaît totalement blanche cristalline, ce qui contredit la couleur des taches caustiques. Prudence, donc, lors de l'utilisation du paramètre `transmit_filter` : pensez à accorder couleur souhaitée et couleur de base du matériau si vous ne désirez pas de `transmit_filter`.

2.2.4 La dispersion

Yafray permet de simuler la décomposition de la lumière blanche au travers d'un prisme ; en pratique, un faisceau de lumière blanche traversant un prisme va être séparé en ses différentes composantes ; en

Figure 10 : À gauche, dispersion_samples = 20, mais sans l'option jitter ; au milieu, le même nombre d'échantillons, mais avec l'option jitter active ; à droite, 100 échantillons et l'option jitter active.

fonction de leur longueur d'onde, les différents rayons ne seront pas réfractés de la même façon et les différentes couleurs, au lieu de se mélanger pour donner de la lumière blanche, vont se séparer en petits rayons colorés, à la manière d'un arc-en-ciel. Le bloc `<shader>` du prisme doit être édité de façon à présenter les attributs suivants :

- `dispersion_power` : définit l'intensité de la dispersion ;
- `dispersion_samples` : définit le nombre d'échantillons à prendre en compte ; en dessous de 10, l'usage de l'option `dispersion_jitter` est fortement recommandé ;
- `dispersion_jitter` : confère à la dispersion une distribution un peu plus aléatoire.

Par exemple, vous pourrez ajouter les attributs suivants à l'un de vos prismes :

```
<dispersion_power value="0.800000" />
<dispersion_samples value="30" />
<dispersion_jitter value="off" />
```

(voir figure 10)

Les reflets du prisme peuvent également être dispersés, révélant sur les objets voisins des taches caustiques aux couleurs de l'arc-en-ciel. Pour obtenir ce type d'effet, il est indispensable de mettre en place une (ou plusieurs) lampes de type Photon pointant sur le prisme.

Conclusion

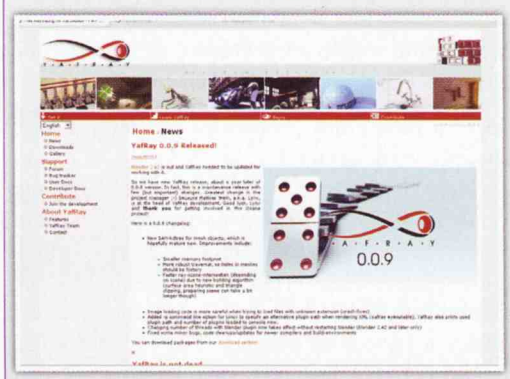
Nous en sommes arrivés au terme de notre voyage au cœur du photoréalisme à l'aide d'un moteur de rendu à la fois puissant et simple d'usage, comme Yafray. Nous avons en effet couvert, quoique non exhaustivement, différents aspects qui vous aideront à donner à vos images cette petite touche supplémentaire qui les rendront singulièrement plus crédibles aux yeux de leurs spectateurs. De l'usage de base de Yafray à la bonne mise en œuvre des shaders, en passant par l'illumination globale ou la simulation d'un effet de flou focal, vous avez désormais toutes les cartes en main pour comprendre et progresser encore plus avec Yafray. Enfin, même s'il s'agit du dernier article de cette mini-série consacrée à Yafray, il n'est pas exclu que nous reviendrons sur ce petit joyau de moteur de rendu (même si son développement n'est aujourd'hui plus très actif, et qu'un *fork* se prépare, avec la bénédiction des développeurs d'origine) dans quelques temps, soit pour présenter d'autres fonctionnalités qui n'ont pas encore été abordées, soit pour présenter des nouveautés, s'il y a lieu d'être.

Olivier Saraja,

olivier.saraja@linuxgraphic.org

LIENS

- Le site de Yafray : www.yafray.org [en]



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

Julien Guellec

A la découverte du protocole de routage OSPF

Cet article a pour objectif de décrire le fonctionnement de l'un des protocoles de routage interne les plus utilisés actuellement : OSPF (Open Shortest Path First). Après un bref rappel de la notion de routage, nous verrons pourquoi OSPF s'impose aujourd'hui comme le protocole de routage de prédilection et comment il peut vous aider à augmenter la fiabilité de votre réseau.

1. Vue d'ensemble

Avant d'aborder le fonctionnement détaillé d'OSPF, il est nécessaire de bien comprendre le rôle fondamental du routage. Cette technique est l'action d'acheminer correctement, et de façon optimale, les paquets à travers différents réseaux. Le routage est effectué par les routeurs et ce sont ces derniers qui implémentent les différents protocoles.

Il existe deux techniques de routage, à savoir le routage dit « statique » et le routage « dynamique ». La différence entre ces deux modes est la façon dont ils acheminent les paquets. Le premier se base sur des routes entrées une à une par l'administrateur réseau et le second utilise un protocole de routage pour déterminer quel est le meilleur chemin à utiliser.

Au sein même des protocoles de routage, nous dénombrons deux familles : les protocoles de routage à vecteur de distance et les protocoles de routage à états de liens. OSPF fait partie de la deuxième famille. Les protocoles de routage à états de liens utilisent l'algorithme du plus court chemin (SPF pour *Shortest Path First*). Contrairement aux protocoles de routage à vecteur de distance (RIPv1, RIPv2 pour les plus connus), les protocoles de routage à états de liens possèdent une vue complète de la topologie du réseau. Ils ont une vue détaillée sur les routeurs distants et les réseaux qui leur sont connectés. Bien entendu, il existe de nombreuses autres différences, que nous allons découvrir au fur et à mesure de cet article, tout en étudiant OSPF.

2. Les origines d'OSPF

Conçu vers la fin des années 80, le protocole OSPF a été développé dans le but de pallier les défauts de RIP. Bien que très simple à implémenter, RIP possède de grosses lacunes, à savoir :

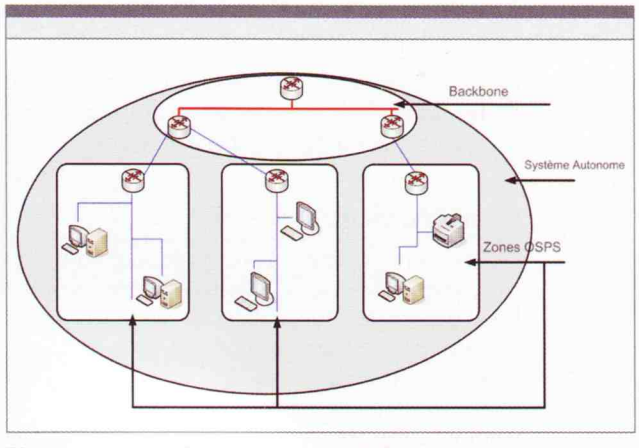
- un temps de convergence élevé (temps au bout duquel tous les routeurs ont une vue cohérente du réseau) ;
- un nombre de sauts limité à 15 qui le cantonne à des réseaux de petites et moyennes tailles (le nombre de sauts est le nombre de routeurs qu'un paquet peut traverser avant d'être considéré comme perdu) ;
- une consommation excessive de bande passante puisque, à chaque mise à jour (par défaut, toutes les 30 secondes), un routeur RIP diffuse la totalité de sa table de routage.

OSPF corrige tous ces défauts, mais présente un inconvénient majeur : celui d'être relativement complexe à mettre en œuvre. En outre, OSPF est également gourmand en termes de ressources matérielles. En effet, de par la complexité des algorithmes mis en jeux (algorithme du plus court chemin), il nécessite plus de ressource CPU et consomme également plus de RAM.

3. Le fonctionnement d'OSPF

3.1 La notion de zones

Un réseau OSPF est divisé en plusieurs zones. Ces zones sont réparties autour d'une unique et même zone fédératrice (la zone 0) également appelée le *backbone*. L'ensemble de ces zones décrit un système autonome (AS pour *Autonomous System*). Plus généralement, un système autonome est le regroupement de plusieurs réseaux implémentant le même protocole de routage interne (IGP pour *Interior Gateway Protocol*).



L'intérêt majeur de la division en différentes zones est de limiter le trafic de routage. En effet, un routeur d'une zone n'aura connaissance que des routeurs se trouvant dans la même zone. Ainsi, sa table de routage sera plus petite, les calculs par l'algorithme SPF en seront diminués, et la convergence accélérée.

Maintenant, il se pose le problème suivant : si les routeurs d'une zone n'ont connaissance que de la topologie de cette même zone, comment peuvent-ils communiquer avec les zones adjacentes ?

3.2 La hiérarchie

Nous venons de voir qu'un routeur OSPF, au sein d'une zone, n'a la connaissance que des routeurs de cette même zone. Pour permettre le dialogue avec les zones adjacentes, OSPF met en place une hiérarchie complexe, et certains routeurs vont devoir accomplir des tâches supplémentaires.

- Le routeur de bordure de zone (ABR pour *Area Boundary Router*) est le routeur connecté au backbone. De par sa position privilégiée, c'est lui qui annonce les routes extérieures à la zone.
- Le routeur de bordure de système autonome (ASBR pour *Autonomous System Boundary Router*) est le routeur chargé de la communication avec les autres systèmes autonomes. Il implémente, en plus d'OSPF, un protocole de routage externe (EGP pour *Exterior Gateway Protocol*) tel que BGP.
- Le routeur désigné (DR pour *Designated Router*) est un routeur élu dans chacune des zones OSPF. Nous verrons par la suite son mode d'élection. Retenons pour le moment qu'il a pour rôle de centraliser l'information de routage au sein de sa zone. Il envoie périodiquement un message d'annonce d'état de lien (LSA pour *Link State Advertisements*) aux autres routeurs de sa zone. Si un routeur quelconque détecte une discontinuité sur le réseau, il en informe le DR, qui se chargera d'inonder le réseau de cette information. Ainsi, les échanges inter-routeurs sont limités, ce qui a pour effet de préserver la bande passante et d'accélérer considérablement la convergence.
- Le routeur désigné de secours (BDR pour *Backup Designated Router*) est, comme son nom l'indique, le routeur qui aura pour rôle de remplacer le DR si celui-ci venait à tomber.

3.3 Les sous-protocoles utilisés par OSPF

Le protocole Hello

Message OSPF de type 1 (HELLO)		
En-tête OSPF avec champ Type à 1 (Message Hello)		
Masque de sous-réseau		
Intervalle HELLO	Options	Priorité du routeur
Intervalle de mort (ou d'arrêt)		
Routeur désigné (DR)		
Routeur désigné de secours (BDR)		
ID du routeur voisin		
ID du routeur voisin		
[...] Éventuellement, d'autres champs ID du routeur voisin peuvent être ajoutés si nécessaire		

Il intervient lors de :

- L'élection du routeur désigné et du routeur désigné de secours. Celle-ci se fait en comparant la priorité des routeurs (par défaut de 1, elle peut être manuellement configurée de 0 à 255). Le routeur ayant la plus grande priorité sera élu DR, le BDR sera le routeur avec la priorité immédiatement plus faible. En cas d'égalité, c'est le champ ID du routeur qui départagera les concurrents (l'ID la plus élevée assure l'élection du routeur). Le DR et le BRD garderont leurs rôles jusqu'à leur défaillance, même si un nouveau routeur avec une priorité plus élevée arrive sur le réseau.
- La vérification de la connectivité entre deux routeurs voisins. Un paquet `Hello` est émis par défaut, et par chaque routeur, toutes les 30 secondes. Dans le cas où un routeur ne recevrait plus de paquets `Hello` de son voisin, pendant un intervalle de temps défini (appelé intervalle de mort), la liaison silencieuse serait déclarée comme rompue, et il en informerait ses autres voisins ainsi que le DR. D'autre part, lors de l'initialisation d'un routeur, celui-ci informe de sa présence et découvre ses voisins via ce protocole.

Les paquets `HELLO` sont émis sur l'adresse multicast 224.0.0.5.

Le protocole d'échange

Il est utilisé lors de l'initialisation des routeurs. Une fois les voisins découverts, nous l'avons vu, grâce au protocole `HELLO`, un routeur doit initialiser sa base de données topologique. Cette initialisation se fait en plusieurs étapes :

- **ExStart** : tout d'abord les routeurs négocient le mode de transfert. Qui sera le maître (*master*) et qui sera l'esclave (*slave*). Cette négociation s'effectue avec des messages OSPF de type 1 (*HELLO packet*)
- **ExChange** : le routeur demande à ses voisins ou au DR de décrire leurs bases de données d'états de liens. Cette demande se fait par un message OSPF de type 2 (DBD pour *DataBase Description packet*) :

Message OSPF de type 2 (DataBase Description packet)		
En-tête OSPF avec champ Type à 2 (Message DBD)		
Interface MTU	Options	Drapeau
Numéro de séquence		
En-tête LSA #1		
[...]		
En-tête LSA #N		

- **Loading** : si le routeur n'a pas connaissance d'une (ou plusieurs) route(s) que l'on vient de lui envoyer, ou si le champ « Âge » est trop vieux, il construit une liste de demandes d'états de liens. Cette liste va servir au routeur pour demander des informations plus précises que celles qu'il vient de recevoir. Cette requête se fait par un message OSPF de type 3 (LSR pour *Link State Request*) :

Message OSPF de type 3 (Link State Request)
En-tête OSPF avec champ Type à 3 (Message LSR)
Type d'état de lien
ID d'état de lien
Routeur annonçant
[...]

La réponse à un LSR est un message OSPF de type 4 (LSU pour *Link State Update*) :

Message OSPF de type 4 (Link State Update)
En-tête OSPF avec champ Type à 4 (Message LSU)
Nombre d'annonces LSA
LSA #1
[...]

Les LSU contiennent eux-mêmes des messages d'annonce de l'état d'une liaison (LSA pour *Link State Advertisement*) :

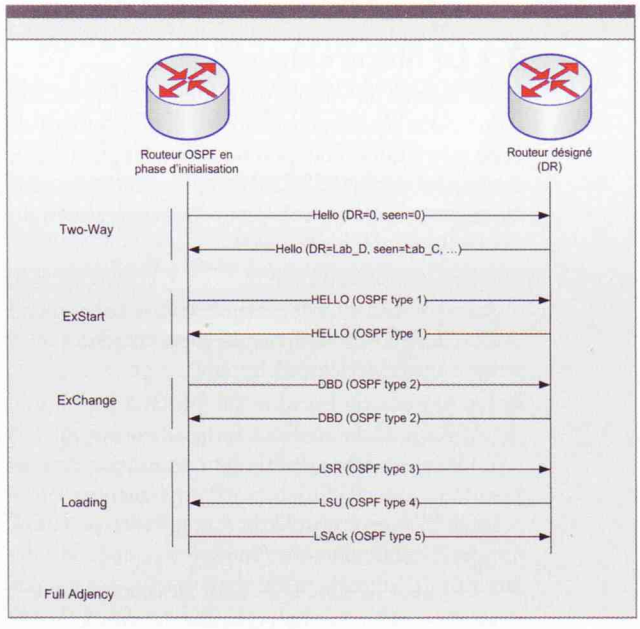
Message OSPF Link State Advertisement (inclus dans un LSU)		
Âge du lien	Options	Type lien
ID d'état de lien		
Routeur annonçant		
Numéro de séquence d'état de lien		
Checksum		Longueur
0 V E B	0	Nombre de liens
ID du lien		
Données du lien		
Type	Nombre TOS	Métrique TOS = 0
TOS	0	Métrique
[...]		
TOS	0	Métrique
ID du lien		
Données du lien		
[...]		

Les LSA sont ensuite acquittés par des messages OSPF de type 5 (LSAck pour *Link State Acknowledgment*) qui reprennent uniquement les en-têtes de chacun d'eux :

Message OSPF de type 5 (Link State Acknowledgment)		
En-tête OSPF avec champ Type à 5 (Message LSAck)		
Âge du lien	Options	Type lien
ID d'état de lien		
Routeur annonçant		
Numéro de séquence d'état de lien		
Checksum	Longueur	
[...]		

▶ **Full Adjency** : (complète adjacence) à la fin du Loading.

Voici un schéma pour clarifier les différentes étapes de l'initialisation d'un routeur OSPF :



Le protocole d'inondation (ou flooding)

Ce protocole est utilisé par un routeur OSPF lorsqu'il détecte un changement d'état de lien. Le routeur en charge de l'annonce (celui qui détecte ce changement) notifie le DR et le BDR de ce changement (sur l'adresse 224.0.0.6). Le routeur désigné inondera la zone dont il a la charge par un LSU contenant l'information. Cette annonce se fait sur l'adresse multicast 224.0.0.5. La réception d'une telle annonce par un routeur OSPF entraîne un nouveau calcul du plus court chemin (SPF).

Conclusion

À travers cet article, nous avons vu qu'OSPF était un protocole relativement complexe, nécessitant non seulement des ressources matérielles importantes, mais également de bonnes connaissances et une bonne maîtrise de son fonctionnement avant de pouvoir être implémenté.

En dépit de cette complexité de mise en œuvre, il augmentera sensiblement les performances, la stabilité et la fiabilité de votre réseau. OSPF est aujourd'hui l'un des protocoles de routage interne les plus utilisés, de par sa force et sa robustesse dont nous venons de parler, mais également grâce à sa licence basée sur des normes ouvertes, qui lui confère un avantage sur des protocoles propriétaires tel que l'EIGRP de Cisco.

Julien Guellec,

SUPINFO Paris,
jguellec@supinfo.com,
http://www.guellec.fr/

LIEN

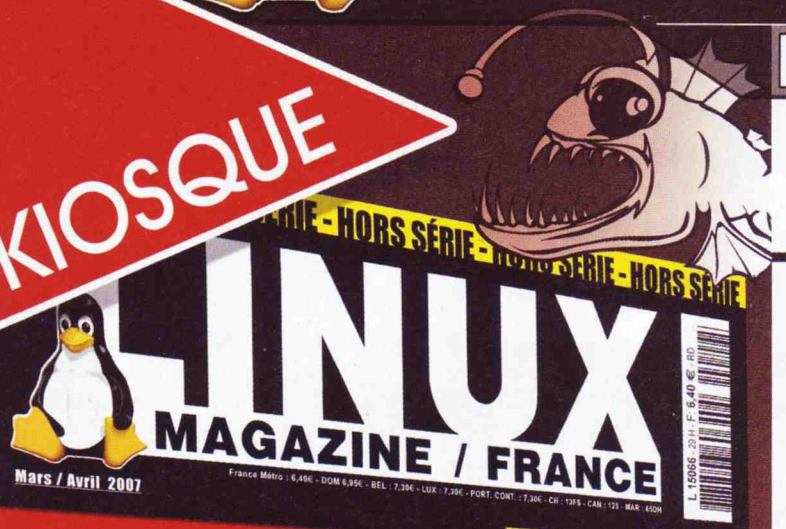
▶ RFC du protocole OSPF :
http://rfc.net/rfc1583.html

Disponible chez votre
marchand de journaux
et sur
http://www.ed-diamond.com

GNU LINUX MAGAZINE / FRANCE

HORS SERIE 29

En KIOSQUE



SOMMAIRE:

100% Pinpin

Cyril Meusnier

Un peu plus loin avec Linux vserver

Dans le numéro 90 de janvier 2007, je vous avais présenté cet outil de virtualisation très utile et rapide à mettre en œuvre. Dans cet article, je vais vous présenter l'aspect plus système et quelques astuces sur les vservers, comme : Comment copier un vserver sur une machine distante ? Comment changer l'adresse IP du vserver sans forcément réinstaller le vserver ?...

1. Introduction

Tout ce qui sera montré ici sont des procédures expérimentées durant un projet lors de mon cursus scolaire. Je vais donc essayer de vous expliquer quelques astuces qui vous aideront sûrement. Par exemple, après avoir réalisé mon projet sous une VMWARE 5 alors qu'il fallait une version 4.5. Là, on se dit : « eh zut, faut que je refasse tout !!!! ». Eh ben, détrompez-vous, une simple copie des bons répertoires suffit. Pour les tests qui vont suivre, et du fait que cela constitue également mon projet, je me placerai dans une distribution Debian Sarge installée dans une VMWARE version 4.5. Mais avant tout cela, il y a une notion à comprendre sur le fonctionnement de vserver qui est celle du contexte. Ensuite, je vous présenterai plus en détail, et avec quelques astuces, ce qu'est un vserver.

2. La notion de contexte

Tout d'abord, maintenant que vous savez installer un vserver, un point important à dire dans un premier temps sur son fonctionnement. Vous vous demandez : mais comment cela marche-t-il ? Je vais essayer d'être très clair. Il faut donc introduire la notion de « contexte ». Le but de ce contexte est de cacher tous les processus et de permettre à un processus externe de ne pas interagir avec un processus d'un vserver. Par défaut, un contexte sur la machine hôte est créé, c'est le contexte « 0 ». C'est lui qui va permettre de définir une séparation entre les différents autres contextes. C'est grâce à la modification du noyau, c'est-à-dire au patch appliqué lors de l'installation, que ceci est possible. Par conséquent, lorsqu'un processus est créé à l'intérieur d'un contexte, ce processus appartient à ce contexte. Faisons par exemple le test de lister tous les processus sur la machine hôte ainsi que ceux appartenant à un vserver. Voyons dans un premier temps le résultat du coté hôte.

```
vmdebian:~# ps a
  PID TTY      STAT   TIME COMMAND
 1934 tty1     Ss+    0:00 -bash
 1935 tty2     Ss+    0:00 /sbin/getty 38400 tty2
 1936 tty3     Ss+    0:00 /sbin/getty 38400 tty3
 1937 tty4     Ss+    0:00 /sbin/getty 38400 tty4
 1938 tty5     Ss+    0:00 /sbin/getty 38400 tty5
 1939 tty6     Ss+    0:00 /sbin/getty 38400 tty6
 1993 pts/0    Ss     0:00 -bash
 2131 pts/1    Ss     0:00 -bash
 2136 pts/1    R+     0:00 ps a
vmdebian:~#
```

On remarque que l'on n'aperçoit que les processus internes à la machine hôte. Pour vous convaincre que c'est vraiment le cas, regardons le résultat à l'intérieur d'un contexte.

```
Loches:/# ps a
  PID TTY      STAT   TIME COMMAND
 1999 pts/0    S+     0:00 /bin/bash -login
 2138 pts/1    S      0:00 /bin/bash -login
 2153 pts/1    R+     0:00 ps a
Loches:/#
```

Le résultat reste quand même différent de ce que l'on a eu dans le premier exemple. En effet, ici nous n'avons que ce qui est interne au contexte, donc bien séparé des autres contextes et donc visible que par le contexte « propriétaire ». Toutefois, une commande existe pour afficher l'ensemble des processus lancés à partir de la machine hôte dont le résultat se trouve sur le troisième exemple. Il s'agit de la commande `vps`.

```
vmdebian:~# vps a
  PID CONTEXT          TTY    STAT  TIME COMMAND
 1934     0 MAIN       tty1    Ss+   0:00 -bash
 1935     0 MAIN       tty2    Ss+   0:00 /sbin/getty 38400 tty2
 1936     0 MAIN       tty3    Ss+   0:00 /sbin/getty 38400 tty3
 1937     0 MAIN       tty4    Ss+   0:00 /sbin/getty 38400 tty4
 1938     0 MAIN       tty5    Ss+   0:00 /sbin/getty 38400 tty5
 1939     0 MAIN       tty6    Ss+   0:00 /sbin/getty 38400 tty6
 1993     0 MAIN       pts/0   Ss    0:00 -bash
 1999 49152 Loches     pts/0   S+    0:00 /bin/bash -login
 2131     0 MAIN       pts/1   Ss    0:00 -bash
 2161     1 ALL_PROC   pts/1   S+    0:00 vps a
 2162     1 ALL_PROC   pts/1   R+    0:00 ps a
vmdebian:~#
```

Pour finir cette explication, sur le schéma de la figure 1, j'ai essayé de faire un résumé de ce que je viens de vous expliquer.

(Voir figure 1, page 49)

J'ai donc essayé de vous montrer ici les différentes interactions possibles entre la machine hôte et le contexte. Dans le vserver 1, j'ai intitulé des processus avec des noms connus. Les flèches montrent que la machine hôte peut voir ces processus. Dans le vserver 2, j'ai nommé deux processus sans nom, juste pour vous montrer que ces deux processus peuvent dialoguer entre eux.

Abonnez-vous !

11 Numéros de Linux Magazine

1 an de bonne lecture, bien UNIX, bien technique... Bref...

Les 3 bonnes raisons de vous abonner !

- Ne manquez plus aucun numéro
- Recevez Linux Magazine chaque mois chez vous, ou dans votre entreprise
- Economisez 15,20 €/an ! (soit plus de 2 magazines offerts !)

- Des offres de couplage sont disponibles
- Retrouvez les Tarifs étrangers hors France Métro sur www.ed-diamond.com

Bon d'abonnement à remplir et à retourner à (ou photocopie)

Linux Magazine - BP 20142 - 67603 Selestat Cedex

11 Numéros de Linux Magazine à 53 € Offre France Métro

Soit Votre Linux Magazine à 4,82 €

(Tarif au numéro dans le cadre d'un abonnement France Métro)

Pour les tarifs étrangers, consultez notre site : www.ed-diamond.com

Les 4 façons de vous abonner !

» Par courrier postal en nous renvoyant le bon ci-dessous.

» Par le Web, sur notre site : www.ed-diamond.com.

» Par téléphone (paiement C.B.) entre 9h-12h & 15h -18h au 03 86 56 02 06.

» Par Fax au 03 86 56 02 09 C.B. et/ou bon de commande administratif

◎ **Oui, je souhaite m'abonner à Linux Magazine pour 11 numéros**

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon réglement :

☐ Je règle par chèque bancaire ou postal à l'ordre de Diamond Editions

☐ Paiement par carte bancaire :

N° Carte :

Expire le : Cryptogramme Visuel : Voir image ci-dessous

Date et signature obligatoire : 200

Offres collectionneurs

LES ANCIENS NUMÉROS !

TOUJOURS DISPONIBLES !

LES 4 FAÇONS DE COMMANDER !

- PAR COURRIER POSTAL EN NOUS RENVOYANT LE BON CI-DESSOUS.
- PAR LE WEB, SUR NOTRE SITE : WWW.ED-DIAMOND.COM.
- PAR TÉLÉPHONE (PAIEMENT C.B.) ENTRE 9H-12H & 15H -18H AU 03 88 58 02 06.
- PAR FAX AU 03 88 58 02 09 C.B. ET/OU BON DE COMMANDE ADMINISTRATIF

BON D'ABONNEMENT À REMPLIR ET À RETOURNER À (OU PHOTOCOPIE)

LINUX MAGAZINE - BP 20142 - 67603 SELESTAT CEDEX

Bon de commande Linux Magazine

RÉFÉRENCE	Prix / N°s	Qté.	Total
Linux Magazine 82 eCos, une autre solution libre pour systèmes embarqués	5,95 €		
Linux Magazine 83 Greylist Eliminez le SPAM à la racine	5,95 €		
Linux Magazine 84 Déploiement de hotspots Wifi sécurisés	5,95 €		
Linux Magazine 85 Firewall: Netfilter & NuFW	6,20 €		
Linux Magazine 86 Serveur SMTP: Routage des mails avec Postfix	6,20 €		
Linux Magazine 87 Le point sur Mono .NET Java et les Brevets	6,20 €		
Linux Magazine 88 Sécurité: Smartcards & Tokens	6,20 €		
Linux Magazine 89 Utilisation avancée de XEN	6,20 €		
Linux Magazine 90 ASTERISK Le serveur de téléphonie IP	6,20 €		
Linux Magazine 91 AJAX AVANCÉ Principe, fonctionnement et pièges	6,20 €		

Bon de commande Linux Magazine Hors Série

LM HS 12 Firewall votre meilleur ennemi Acte 1	5,95 €		
LM HS 13 Firewall votre meilleur ennemi Acte 2	5,95 €		
LM HS 15 GIMP et la Photo	5,95 €		
LM HS 16 Kernel (1)	5,95 €		
LM HS 17 Kernel (2)	5,95 €		
LM HS 18 Haute Disponibilité	5,95 €		
LM HS 19 The Gimp 2.0	5,95 €		
LM HS 20 PHP 5	5,95 €		
LM HS 21 Recyclez vos PC	6,40 €		
LM HS 22 GIMP et le Web	6,40 €		
LM HS 23 Linux et electronique	6,40 €		
LM HS 24 Linux Embarqué	6,40 €		
LM HS 25 Linux Embarqué 2	6,40 €		
LM HS 26 Spécial The GIMP	6,40 €		
LM HS 27 Électronique et Linux	6,40 €		
LM HS 28 Administration systeme avec Debian	6,40 €		
	sous TOTAL		
	Frais de port France Metro		+ 3,81 €
	Frais de port Etranger		+ 5,34 €
	TOTAL		

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

GNU LINUX MAGAZINE / FRANCE

2 Je joins mon réglement :

☐ Je règle par chèque bancaire ou postal à l'ordre de Diamond Editions

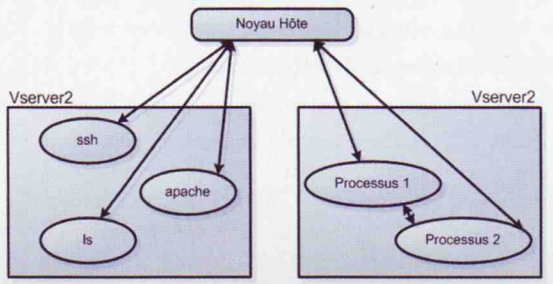
☐ Paiement par carte bancaire :

N° Carte :

Expire le : Cryptogramme Visuel : Voir image ci-dessous

Date et signature obligatoire : 200

Figure 1 : Schéma récapitulatif



2.1 Création d'un contexte

Lors de la création d'un nouveau contexte, ce dernier va être associé à un sous-répertoire de la machine hôte. Ce répertoire va ensuite correspondre à la « racine » du vserver ou plutôt du contexte. Il contiendra tous les services, répertoires ou encore fichiers nécessaires pour le bon fonctionnement du vserver. Au final, nous avons un ensemble de processus tournant dans un endroit « clos » disposant d'un configuration personnelle : nom de machine, adresse IP, utilisateur... et cela constitue un serveur virtuel.

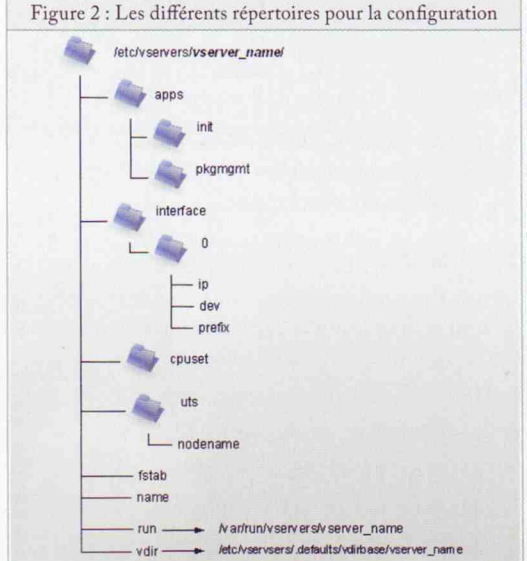
2.2 Le réseau dans tout cela...

Chaque contexte dispose d'une adresse IP. Le noyau fera un routage vers le processus adéquat contenu dans le bon contexte. Il est donc impossible que deux vservers disposent de la même adresse IP. Cela constituerait un conflit d'adressage IP, comme ce serait le cas dans un LAN avec deux adresses identiques.

3. Qu'est-ce qu'un vserver ?

Vous avez sans doute déjà installé votre vserver sans comprendre le rôle de chaque répertoire. Je vais vous expliquer l'utilité de certains d'entre eux. Tout d'abord, après avoir suivi le processus d'installation et créé votre premier vserver, vous aurez une architecture comme représenté sur la figure 2.

Figure 2 : Les différents répertoires pour la configuration



Dans ces répertoires, il va vous être possible de réaliser plusieurs choses. Je vais vous expliquer comment faire en sorte qu'un vserver démarre automatiquement au démarrage du poste principal ou encore comment faire pour que, lorsque l'on lance la commande `ifconfig`, nous arrivions à voir apparaître l'adresse IP de la machine.

3.1 Démarrer un vserver au démarrage de la machine hôte

Lors de la machine hôte, il est possible de choisir de démarrer un vserver automatiquement. Dans la plupart des documentations vues sur le net, dans le fameux fichier `vserver_name.conf`, il suffisait de rajouter `ONBOOT=yes`. Or, depuis le changement, il faut pour cela faire la chose suivante :

```
# mkdir -p /etc/vservers/vserver1/apps/init
# echo "default" > /etc/vservers/vserver1/apps/init/mark
```

Dans le répertoire `apps/init` correspondant à notre vserver, nous allons créer un fichier contenant juste `default`. Au démarrage, un service va donc détecter ce fichier et lancer tous les vservers qui auront ce fichier.

3.2 Connaître l'adresse IP d'un vserver en étant dans le vserver

Il est très difficile de vous présenter toutes les options possibles des Linux vserver. Dans l'ancienne version, il suffisait d'éditer un fichier du genre `vserver_name.conf` dans le répertoire `/etc/vservers`. Depuis, les choses ont pas mal évolué. Dans le répertoire d'un Linux vserver, il est possible de créer des fichiers de configuration bien spécifique. Par exemple, lorsque vous êtes dans un vserver, vous allez sûrement trouver dommage de ne pas connaître en tapant `ifconfig` l'adresse IP du serveur. Pour ce faire, arrêter le vserver, éditer le fichier `/etc/vserver/vserver_name/interface/0/name` et mettez un nom, par exemple `dummy`. Relancez votre vserver, tapez `ifconfig` et là vous aurez l'adresse IP du serveur.

```
Loches:/# ifconfig
eth0      Lien encap:Ethernet  HWaddr 00:0C:29:E4:97:59
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2190 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1912 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          RX bytes:289857 (283.0 KiB)  TX bytes:678615 (662.7 KiB)
          Interruption:18 Adresse de base:0x1080

eth0:dumm Lien encap:Ethernet  HWaddr 00:0C:29:E4:97:59
          inet adr:192.168.0.21  Bcast:192.168.0.255
Masque:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interruption:18 Adresse de base:0x1080

Loches:/#
```

3.3 Exemple d'un fichier pris sur le net et réalisation avec la nouvelle méthode

Dans cette partie, nous allons prendre exemple sur un fichier pris sur le net et voir comment faire la même chose, mais avec la nouvelle façon de faire. Nous avons bien essayé de faire avec un fichier de même type mais, hélas, cela n'a pas donné le résultat tant espéré.

```
# Most easy thing is to have an own ip-address for each vserver
IPROOT=10.95.81.17
IPROOTMASK=255.255.255.0
# How shall the networkdevice be named from the view of the
vserver
IPROOTDEV=eth0
# shown hostname
S_HOSTNAME=mysql5
# lock = you are not allowed to create a new context in your
context
# nproc = let ULIMIT-value be global for this context
S_FLAGS="lock nproc"
# What capabilities shall the server have? Look for explanation
S_CAPS="CAP_NET_RAW CAP_NET_BIND_SERVICE"
# Start this server on booting? This will be checked in /etc/init.
d/vservers
ONBOOT=yes
```

Dans la première partie de ce fichier, nous trouvons l'adresse IP, le masque du vserver et le nom vu par le vserver pour l'interface réseau. Allez dans le répertoire `/etc/vservers/vserver_name/interface/0/`. Vous trouverez 3 fichiers.

- `ip` : contenant l'adresse IP ;
- `prefix` : le masque de sous-réseau ;
- `dev` : contenant le nom du périphérique réseau.

Il vous suffit donc de remplir ces trois fichiers pour configurer l'interface réseau. `S_HOSTNAME` correspond au nom de la machine. Ce dernier se trouve dans `/etc/vservers/vserver_name/uts/nodename`. Inscrivez dans ce fichier le nom que vous désirez, relancez et constatez le changement. Le fichier `name` se trouvant dans le répertoire du vserver contient le nom du vserver. La ligne `S_FLAGS` sera désormais dans le fichier `flags` directement dans le répertoire du vserver. Au même endroit que pour les ***flags***, mais dans le fichier `capabilities`, vous mettrez la ligne `S_CAPS="CAP_NET_RAW CAP_NET_BIND_SERVICE`. La dernière ligne a déjà été traitée dans le paragraphe « 3.1 Démarrer un vserver au démarrage de la machine hôte ».

3.4. Le déplacement d'un vserver sur une machine distante

Dans cette partie, je vais vous expliquer comment copier un vserver sur une autre machine. Mais avant tout, il est bon de connaître l'architecture d'un vserver. Un vserver est une sorte de « ***chroot*** amélioré ». On peut changer pas mal de choses sur un vserver, si on connaît bien les répertoires qui le composent ainsi que les fichiers dans ces répertoires. Si vous avez bien suivi la méthode d'installation de l'article précédent, vous trouverez donc dans votre système deux répertoires importants. Dans le premier, `/var/lib/vservers/vserver_name`, vous trouverez l'architecture du vserver. C'est-à-dire tous les répertoires de votre système, ce que vous voyez lorsque vous entrez dans votre vserver. Le deuxième, `/etc/vservers/vserver_name`, présenté sur la figure 2, contient les fichiers de configuration de votre vserver ainsi qu'un lien vers le premier répertoire présenté. La commande reste simple, car elle consiste en une synchronisation entre les deux machines,

```
#rsync -e ssh -avHl /vservers/vserver_name
adresse_ip_machine_distante:/etc/vservers/vserver_name
#rsync -e ssh -avHl /var/lib/vservers/vserver_name
adresse_ip_machine_distante:/var/lib/vservers/vserver_name
```

puis à tester en lançant sur la machine distante si le vserver a bien été synchronisé.

3.5 Listes des outils

La liste des commandes qui vont vous êtes présentées ne sont utilisables qu'à partir du contexte 0, c'est-à-dire à partir de la machine hôte.

- `vtop` : `top` global sur tous serveurs ;
- `vps` : `ps` global sur tous serveurs ;
- `vpstree` : `pstree` global sur tous serveurs ;
- `vdu` : `du` qui ne se mélange pas avec les liens ;
- `vkill` : `kill` global sur tous serveurs ;

- `vserver-copy` : copie de vservers ;
- `vfiles` : extraire la liste des fichiers spécifiques (les liens durs) à un vserver (gain de place) ;
- `vunify` : inclure les fichiers manquants (liens symboliques) après un `vfiles`.

4. Caractéristiques d'un vserver

Pour ceux qui n'auraient pas lu l'article du mois dernier, je vais vous réexpliquer certaines caractéristiques d'un vserver. Tout d'abord, la première chose, c'est que comparé à d'autres méthodes comme VMWARE qui, lui, utilise la totalité des ressources de la machine, vserver, n'utilise que ce dont il a besoin. Pour faire plus simple, ce qui est utilisé en ressource par un vserver correspond aux processus tournant à l'intérieur.

Ce système reste un système assez sécurisé. Comme présenté dans la figure 1, on remarque que les discussions entre vservers sont totalement impossibles. De plus, même un super-utilisateur ne peut voir ce qui se passe à l'extérieur de son contexte. Dans le cas d'une intrusion, la seule partie visible sera le contenu du contexte correspondant.

La maintenabilité d'un tel système reste simple. Il suffit de faire régulièrement des sauvegardes des répertoires du vserver dans, par exemple, un fichier `tar.gz` et de le décompresser quand cela est nécessaire, puis ensuite de relancer le vserver et tout marche à nouveau.

L'un des seuls inconvénients que l'on puisse trouver, c'est la place occupée par chaque vserver. Comme chacun dispose de son propre système de fichiers, que chaque programme se situe dans un vserver, au final, la place disque sera de plus en plus petite. En moyenne, les systèmes que j'ai mis en place pèsent environ 400 Mo. Imaginez la place prise lorsque, sur une machine en production, il y a 10 vservers qui tournent. Une technique d'unification existe pour permettre à plusieurs vservers de se partager des applications, ce qui permet un gain considérable. Comptez environ 600 voire 700 Mo au lieu de 4 Go correspondant à la place occupée par les 10 vservers sans unification.

Conclusion

Linux vserver est donc un système de virtualisation très performant et sécurisé à mon sens. La sauvegarde d'un tel système est très simple à faire et la maintenabilité simplifiée. Si une partie est affectée, la restauration se fait rapidement. Le système peut être gourmand en espace disque, mais non en ressource. L'une des seules limites que l'on peut constater, c'est que chaque vserver se partage le même noyau. Dans la plupart des cas, cela ne constitue pas une gêne importante, sauf dans le cas où une application nécessite un patch du kernel.

Cyril Meusnier,

cyril.meusnier@etu.univ-tours.fr

Yann Allandit

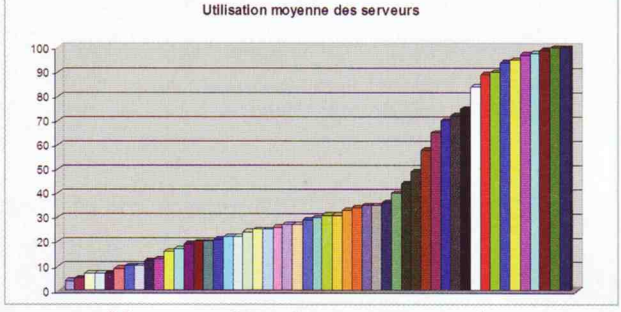
Problématique de consolidation et atteinte des objectifs de niveau de service (SLO) avec Xen

Ou comment gérer les priorités applicatives entre plusieurs environnements partagés au sein d'un même serveur.

1. Défis de la [para]virtualisation dans des contextes de consolidation

De nombreuses entreprises s'interrogent sur l'optimisation de leur infrastructure et de leurs investissements informatiques. En effet, les restrictions budgétaires imposent une limitation du nombre de serveurs, ainsi qu'une remise en cause de la logique dite « des silos » où un serveur héberge une application. Historiquement, les serveurs étaient configurés en fonction des pics de charges applicatives. Ce modèle architectural entraîne généralement une sous-utilisation des capacités de traitement, mais aussi parfois une saturation de celles-ci. Dans la figure 1, nous proposons un exemple de taux d'utilisation CPU moyen par serveur (extrait d'un cas réel). On constate qu'une grande majorité de ces machines dispose de capacités de traitement dont l'entreprise en question ne profite pas. Une étude récente a d'ailleurs montré que le taux moyen d'utilisation d'un serveur « Unix » était de 30% alors que pour « Windows » ce taux tombait à 15%.

Figure 1 : Utilisation moyenne des CPU dans une entreprise pilote



Alors, que faire de ces capacités de traitement inutilisées ?

La réponse à cette question se trouve en partie dans les projets de consolidation qui, pour nombre d'entre eux, reposent sur des technologies de virtualisation.

Ces technologies peuvent être des suites complètes de type *adaptive infrastructure* basées sur des composants matériels et logiciels comme HP *Virtual Server Environment* ou IBM *Virtualization Engine*.

Elles peuvent aussi être complètement virtualisées au niveau logiciel en offrant, dans ce cas, des fonctionnalités moindres. La référence dans ce domaine étant ESX de VMWare. Mais, il existe d'autres solutions comme l'*Integrity Virtual Machines* de HP ou encore Xen qui servira de support à cet article.

Très concrètement, ces solutions de virtualisation ou de paravirtualisation permettent de faire fonctionner plusieurs serveurs logiques sur une même machine physique (*host*). Les deux principaux défis de telles solutions sont, d'une part, l'isolation et la sécurité des machines virtuelles les unes par rapport aux autres. L'octroi de ressources et la gestion des priorités entre les différentes machines virtuelles (VM), d'autre part. Ce point est d'autant plus critique qu'il conditionne l'atteinte des objectifs des niveaux de service (SLO pour *Services Level Objectives*). C'est ce deuxième point que nous allons détailler dans cet article au travers d'un exemple de gestion de charge et d'allocation de ressources. Mais commençons par un rappel de ce qu'est Xen et de ses principes de fonctionnement.

2. Xen : quelques rappels

Xen **[1][2]** est un Moniteur de Machines Virtuelles (VMM) dont certains aspects ont été présentés dans les *Gnu/Linux magazine* 85, 87 et 89. Xen est une technologie dite « de paravirtualisation » dans le sens ou une machine virtuelle de type Xen nécessite une modification de son noyau. On utilise parfois dans ce cas le néologisme « Xenification » pour parler d'un OS modifié de la sorte.

Une machine virtuelle Xen est appelée « Domaine ». La première d'entre elles est le Domain-0 en référence au numéro d'identification affecté à chaque domaine actif. Ce Domain-0 est celui qui est créé lors du *boot* du serveur. Aux yeux d'un utilisateur, ce domaine initial est similaire à une machine standard. Il est cependant différent dans la mesure où il a la charge de la gestion et de la communication avec les autres domaines instanciés sur le serveur.

Les domaines utilisateurs, généralement appelés « domaineU », se caractérisent principalement par un fichier de configuration fournissant de nombreuses informations comme le nom, la mémoire allouée, le nombre de CPU virtuelles (appelées par la suite « VCPU » dans cet article), un identificateur universel unique (UUID), le pont réseau, l'amorce de démarrage, ainsi que de nombreuses autres informations optionnelles. Une machine virtuelle Xen se compose également d'un espace de stockage (qui peut être un fichier, un disque, une partition physique ou encore un volume logique).

3. Problématique de la gestion de la charge et des SLO

Lorsque plusieurs applications cohabitent sur un même système, se pose la question de l'allocation des ressources, du partage des capacités de traitement ainsi

que de la gestion des priorités. Dans un environnement multidomaine Xen, l'accès aux capacités de traitement repose sur un ordonnanceur (*scheduler*) qui séquence et équilibre la charge générée par les VCPU des systèmes invités entre les CPU disponibles du système hébergeur multiprocesseur. Distinguons ici les ordonnanceurs de type préemptif qui assignent un client à une CPU en fonction de différents types d'événements (une VCPU est bloquée, cède sa place, a consommé son temps ou est réveillée) et les non préemptifs qui n'exécuteront une décision d'ordonnancement que lors de la libération volontaire d'une CPU par un client actif. Le mode préemptif est particulièrement adapté dans des contextes de fortes charges (e/s et CPU) ainsi que dans des environnements multiprocesseurs.

Xen est singulier dans le monde de la virtualisation dans le sens où il permet de choisir entre 3 ordonnanceurs différents. Charge à l'administrateur système d'opter pour le mieux adapté à ses besoins. Le choix devra se faire entre :

- *Borrowed Virtual Time* (BVT). Basé sur le concept de temps virtuel, ce scheduler classe les VM en fonction de leur temps d'exécution supposé. Il exécute celle ayant le temps le plus court en premier. Il était utilisé avec Xen version 2.
- *Simple Earliest Deadline First* (SEDF). Le SEDF utilise un algorithme temps réel pour assurer du temps garanti. Il dédie une portion de temps global définissable à un domaine et permet de dire si oui ou non le domaine peut excéder le temps CPU qui lui a été dédié initialement (en cas de non saturation du système).
- *Credit Scheduler*. Il est le plus récent des schedulers. Il permet de gérer automatiquement l'équilibrage de charge sans intervention de l'administrateur. L'étude réalisée ici se base sur ce modèle. Nous allons donc expliquer en détail son fonctionnement.

Pour connaître le type de scheduler actif sur un système, l'option `dmesg` (comme *diagnostic message*, une commande OS qui imprime les messages du *buffer* de noyau) de la commande `xm` devra être utilisée :

```
XEN# xm dmesg | grep -i sched
(Xen) Using scheduler: SMP Credit Scheduler (credit)
```

La sélection d'un scheduler se fera en modifiant le fichier `grub.conf` et en ajoutant le paramètre `sched=XXX`. `XXX` correspondant au mode d'ordonnancement retenu. Toutefois, même s'ils restent disponibles pour le moment, il est prévu de désactiver voire de supprimer BVT et SEDF du noyau Xen. Il est donc fortement conseillé d'utiliser le Credit Scheduler.

Du point de vue algorithmique, chaque CPU physique gère sa propre file d'attente de VCPU. Cette file d'attente est triée selon le niveau de priorité de chacune des VCPU. La priorité d'une VCPU peut être de deux types, *over* ou *under*, c'est-à-dire au-dessus ou au-dessous. Ce statut caractérise les VCPU qui ont ou n'ont pas dépassé leur temps d'accès (en anglais, on parle de *fair share*) aux ressources CPU sur la période courante. Très logiquement, lorsqu'une VCPU est ajoutée à une file d'attente, elle est placée après les autres VCPU de priorités similaires.

Quand une VCPU est active, elle consomme des crédits. La valeur atomique est appelée le *tick*, elle est de 10 ms. À intervalles réguliers – toutes les 30 ms – un *thread* système recalcule les crédits qu'une VM active a gagnés et les alloue. Une valeur négative indique une priorité de type over. Tant qu'une VCPU n'a pas consommé ses crédits, sa priorité est de type under.

Chaque CPU, à chaque décision d'ordonnancement, activera la VCPU ayant un statut under et se trouvant au sommet de la file d'attente. La décision de scheduling fait partie intégrante de l'activité de l'ordonnanceur. Il est par conséquent prévu pour être léger et efficace. Il est à noter que le Credit Scheduler a été écrit pour assurer une meilleure efficacité dans les gros systèmes symétriques multiprocesseurs (SMP) que BVT et SEDF qui ont de réels problèmes dans ce genre d'environnements.

Si une CPU ne trouve pas de VCPU ayant une priorité de type under dans sa file d'attente locale, elle ira chercher dans les autres files. Une CPU ne passera en statut inactif (*idle*), que si aucune autre VCPU n'a de tâche à exécuter (aucune VCPU en attente, quel que soit son statut). Ce système d'équilibrage de charge (*load balancing*) garantit que chaque VM disposera d'un temps d'accès aux ressources CPU équitable.

Il est cependant possible de désactiver cet équilibrage de charge en affectant une VCPU particulière à une CPU physique et ainsi restreindre l'usage de celle-ci en utilisant la fonction générique `vcpu-pin`. Notons que l'affectation d'une VCPU à une CPU n'est pas exclusive. C'est-à-dire que la CPU pourra traiter des tâches soumises par des VCPU d'autres domaines. Pratiquement, le temps total de traitement va être réparti entre les différentes VM.

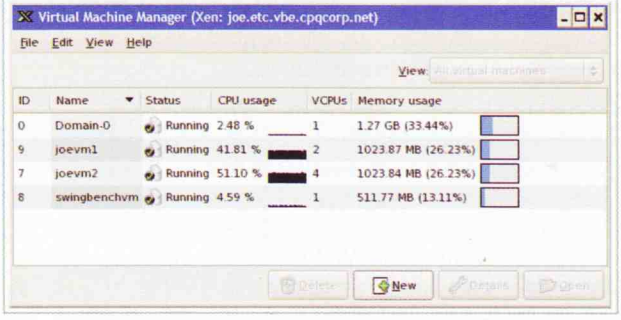
Dans l'exemple ci-dessous, nous affectons la VCPU 0 du domaine `swingbenchvm` à la CPU 3 de la machine hôte. Avant cela, la commande `xm list` fournit la liste des domaines actifs sur le serveur. Ceci permet d'identifier les noms et identifiants des domaines. Ces informations sont nécessaires à l'utilisation de la fonction `vcpu-pin`.

```
Usage: xm vcpu-pin <Domain> <VCPU> <CPU>
XEN# xm list
Name                              ID Mem(MiB) VCPUS State   Time(s)
Domain-0                           0     1817     1 r-----  24974.0
joevm1                             9     1024     2 r-----  71304.3
joevm2                             7     1024     4 -b----  11381.7
swingbenchvm                       8      512     1 -b----  33227.4
XEN# xm vcpu-pin swingbenchvm 0 3
```

Les outils `xentop` (mode texte) ou `virt-manager` **[3]** (mode graphique) délivrent une image instantanée de l'activité globale du système hôte.

```
XEN# virt-manager
```

Figure 2 : Répartition du temps CPU entre les différentes VM – consommation instantanée plus historique



La gestion des charges et donc des priorités applicatives nécessaires à l'atteinte des SLO s'appuiera sur l'algorithme du Credit Scheduler. De plus, un administrateur pourra, manuellement, pondérer une VM. Il pourra aussi limiter le temps qu'une CPU consacre à un domaine (on parle alors de *caping*). Pour cela, la commande `xm sched-credit`, dont nous donnerons plusieurs exemples par la suite, sera utilisée.

Présentons maintenant ces fonctions de caping (ou activation de seuil supérieur) et de pondération.

4. Le caping de l'activité CPU

D'une manière générale, un ordonnanceur peut fonctionner selon les deux modes : *work-conserving* et *non work-conserving* (NWC). Dans le premier mode, une CPU est considérée comme libre si aucune tâche associée à une VCPU, quel que soit le domaine d'origine, n'est en attente sur le système hébergeur.

A contrario, en mode « NWC », le temps CPU dédié à un domaine ne pourra excéder la part initialement allouée. On parlera alors de seuil supérieur. Le caping est donc l'option permettant de limiter le montant maximum de temps CPU qu'un système invité va être capable de consommer, même si le système hôte dispose de cycles CPU non utilisés. L'activation du caping permet ainsi de passer du mode work-conserving (mode par défaut de Xen avec le Credit Scheduler) au mode NWC.

Ce seuil supérieur est exprimé en pourcentage d'utilisation d'une CPU physique. 100 est donc 1 CPU physique, 50 est la moitié d'une, 400 est l'équivalent de 4 CPU, etc. La valeur par défaut, 0, signifie qu'aucun seuil n'a été défini pour une VM.

Dans l'exemple ci-dessous, le caping d'une VM disposant de 4 VCPU a été activé. Au temps 0, la VM consomme l'ensemble des ressources disponibles puisqu'il n'y a ni compétition, ni limite de définie. Dans le temps 1, nous limitons à 50% la consommation CPU maximum autorisée pour le domaine `joevm2`. Le seuil supérieur étant exprimé en pourcentage global, il aura une valeur de 200 (50%x4). On constate alors qu'approximativement 50% des ressources CPU sont inutilisées ou réservées aux autres domaines.

Impact du caping sur le temps CPU consommé par un domaine

Temps 0	Temps 1
`XEN# xm sched-credit -d joevm2 {'cap': 0, 'weight': 256}`	`XEN# xm sched-credit -d joevm2 -c 200`

Enfin, précisons que quel que soit le nombre de VCPU, à caping identique, deux VM consommeront très sensiblement la même charge CPU. Par exemple, une VM disposant de 4 VCPU limitées à 25% chacune (soit un total de 100) disposera des mêmes ressources que 2 VCPU limitées à 50% l'unité. Nos tests ont confirmé cette évidence.

5. Pondération des VM

La pondération se distingue de la logique de seuil dans la mesure où elle ne s'attache pas à limiter la consommation CPU d'une VM, mais plutôt à valoriser l'importance d'un environnement par rapport à un autre. De plus, la pondération n'est pas une option, puisque tout domaine Xen a un poids dès sa création.

Ce poids s'exprime au travers d'un entier qui aura une valeur comprise entre 0 et 65535. Plus le poids est élevé, plus le temps CPU consacré au domaine sera important. Un domaine avec un poids de 512 aura deux fois plus de temps CPU qu'un domaine avec un poids de 256. A condition bien entendu qu'il y ait de la contention sur le système. Il est conseillé de pondérer une VM avec une valeur multiple de 256 qui constitue la valeur par défaut. À chaque recalcul de crédit par le scheduler – toutes les 30 ms – ce poids sera pris en compte et aura pour effet de ralentir ou d'accélérer le passage d'une VCPU du statut under à celui de over et donc à privilégier une VM plutôt qu'une autre.

La ligne de commande suivante permet de modifier le poids d'un domaine :

```
XEN# xm sched-credit -d mon-domaine -w 512
```

`-d`, pour *domain*, est le paramètre pour l'identifiant du domaine et `-w`, pour *weight*, celui du poids affecté.

Maintenant que ces concepts sont définis, il est temps de les évaluer. C'est ce que nous nous proposons d'étudier dans la deuxième partie de cet article.

6. Création d'un environnement de test

Nous l'avons dit précédemment, la gestion de la charge nécessitera un arbitrage uniquement en cas de contention sur le serveur physique. Si l'usage des CPU n'est pas saturé, il n'y a nul besoin d'intervenir, puisque chaque domaine dispose des ressources dont il a besoin.

Dans ce contexte de consolidation avec Xen ou tout autre outil de virtualisation, il sera également judicieux de faire cohabiter une application critique soumise à SLO avec une ou plusieurs autres applications moins sensibles. Ceci permettra de définir aisément les priorités.

C'est dans cet esprit que nous avons créé un environnement de test capable de générer suffisamment de charge CPU pour démontrer les fonctionnalités de caping et de pondération. Cet environnement, se voulant le plus réaliste possible, comprend une application de type transactionnel et une autre de type décisionnel.

La figure 3 décrit les machines virtuelles créées pour l'occasion. En plus du *domain-0*, deux applications réparties entre trois domaines se partagent les ressources du host. D'un coté, se trouve `hammerora` **[4]**, application de type décisionnelle consommatrice intensive de CPU et d'accès disque. De l'autre, l'environnement OLTP nommé `swingbench` **[5]**. Dans le détail, `hammerora` est complètement intégré au sein du même domaine (`joevm2` dans la figure 3) alors que `swingbench` est réparti en deux domaines qui, très prosaïquement, représentent un serveur d'application et un serveur de base de données (`joevm1`). La communication se faisant par le pont réseau interne.

Dans les deux cas, nous avons retenu Oracle 10g comme support de base de données. Il est à noter qu'Oracle supporte l'utilisation de sa base de données avec Xen. Cependant, si les conseils du support Oracle ne permettent pas de résoudre un problème identifié, celui-ci devrait être transféré au support Xen. S'il s'avère que le problème est bien lié à la base de données, il sera alors demandé au client de reproduire celui-ci dans un environnement natif (non paravirtualisé).

Du point de vue matériel, un serveur équipé de 4 CPU de type Intel Xeon avec 4GB de ram héberge ces domaines.

Figure 3 : Description de l'environnement de test

Avant de passer au contenu et aux résultats des tests, présentons succinctement les générateurs de charge `swingbench` et `hammerora`.

`Swingbench` est un générateur de charge de type *freeware*, c'est-à-dire sans licence ni coût de support. Il présente l'avantage de fonctionner avec des bases mono-instances et en grappes (connu sous le nom de RAC). Son interface graphique permet de visuellement mesurer la performance de l'environnement testé.

Le *benchmark* (banc de test) fourni se base sur le schéma `oe` qui est livré en standard comme modèle avec les bases de données. Il peut être utilisé sans interruption

jusqu'à ce que l'espace disque dédié au schéma soit complètement saturé. Swingbench introduit une forte contention sur un petit nombre de tables.

L'activité générée se compose d'un mix de 5 types de transactions associant des ordres de lecture (*select*), de mise à jour (*update*) et d'ajout (*insert*) de données. Ceci garantissant un équilibre entre les ordres de lecture et d'écriture.

Le *framework* complet a été développé en Java et peut, par conséquent, être utilisé sur une grande variété de plateformes.

Hammerora, quant à lui, est un outil du monde libre pour les bases Oracle 8i, 9i et 10g écrit en TCL/TK. Il nécessite l'installation du client Oracle pour fonctionner.

Hammerora transforme des fichiers traces (de requêtes SQL, PL/SQL ou encore procédures stockées) en programme tcl.

Le tcl présente l'avantage d'offrir des interactions avec la base de données sans l'inconvénient d'avoir à recompiler les programmes générés. Il s'appuie pour cela sur les interfaces d'appels Oracle (OCI) **[6]**. Chaque programme utilisé ici est compatible tcl/oratcl et peut, par conséquent, être exécuté indépendamment d'Hammerora en ligne de commande TCL *shell*.

L'outil permet donc de simuler de multiples utilisateurs accédant à une base pour y exercer une action. Le nombre de connexions virtuelles et le nombre d'itérations sont paramétrables.

En standard, Hammerora fournit un schéma et des requêtes tpc-c (mode transactionnel donc). Nous avons, dans le cadre de ce test, préféré développer un environnement spécifique composé d'un schéma et d'une requête de type *datawarehouse* qui se trouve ci-dessous.

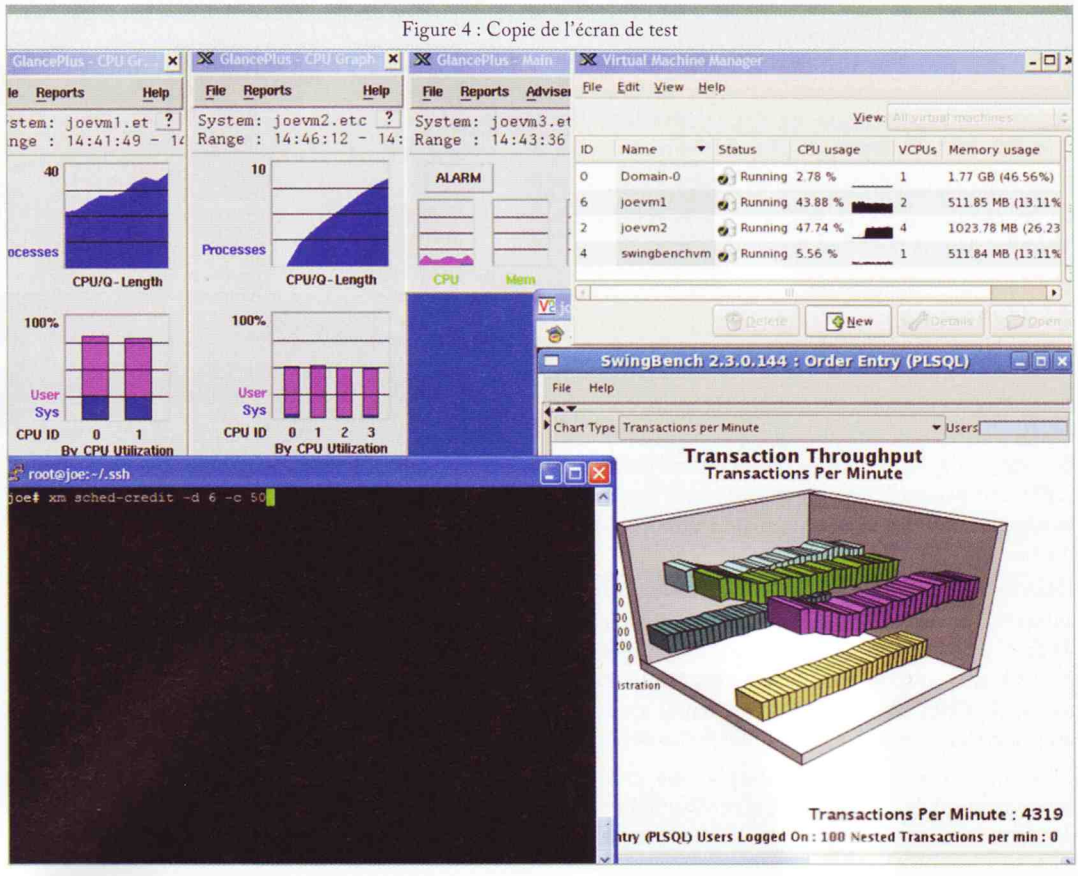
```
select l.region,sum(si.quantity) qty_sold
from locations l,warehouses w, inventories i,products
p,sale_items si
where l.location_id=w.location_id
and w.warehouse_id = i.warehouse_id
and i.product_id = p.product_id
and p.product_id = si.product_id
group by l.region
```

Environ deux secondes sont nécessaires pour exécuter cette requête. Les tables les plus importantes sont locations et sale_items avec respectivement 1946 et 534624 enregistrements.

La métrique retenue pour évaluer l'impact du caping et de la pondération sur les performances est le nombre de transactions par minute générées par swingbench.

La figure 4 représente l'interface des tests. Des outils comme GlancePlus **[7]** ou Xosview **[8]**, permettent de visualiser graphiquement l'activité système de chacune des VM.

Figure 4 : Copie de l'écran de test



7. Résultats

Les problématiques d'atteinte de SLO dans les environnements consolidés et les outils de virtualisation ont servi de fil rouge à cet article. Comme nous l'avons vu, Xen fournit des environnements applicatifs isolés et sécurisés. Il fournit aussi des options de gestion de charge détaillées précédemment.

Grâce à l'environnement de test, nous avons maintenant la possibilité de mesurer les écarts de performances et ainsi prouver le bon fonctionnement de ce concept (tableau suivant et figure 5).

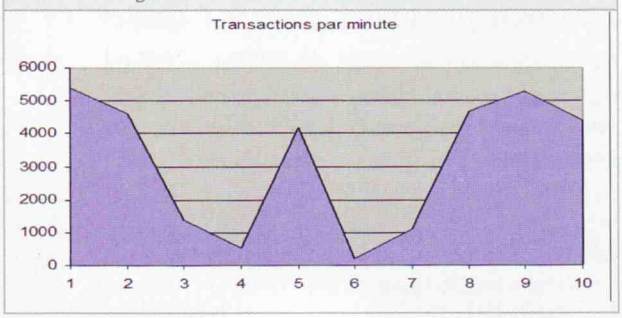
Les étapes 1 et 2 montrent l'activité transactionnelle sans caping et à pondération identique. À l'étape 1, seul l'environnement OLTP – `swingbench` – est actif. À l'étape 2, `hammerora` est démarré et entre en compétition avec `swingbench`. Puis les VCPU de l'environnement de base de données OLTP sont bridées (étapes 3 et 4) avant que l'option de caping ne soit désactivée (étape 5). Dans un deuxième temps, le poids de la VM `hammerora` est considérablement augmenté (64 fois supérieur (étape 6) puis 8 fois supérieur (étape 7) à la valeur par défaut). A l'étape 8, l'équilibre entre les pondérations est rétabli grâce à l'augmentation du poids de la VM `swingbench`. Pour finir, les deux VM sont ramenées à leurs paramètres initiaux (étape 9 et 10).

Tableau de correspondance entre les actions de scheduling et l'impact sur les performances

#	Domain id	Action	Tpm result
1	2, 4, 6	-c=0 -w=256 for all; startup swingbenh; hammer down	5400
2	2, 4, 6	Startup hammerora	4600
3	6 (swb db)	-c=50	1400
4	6 (swb db)	-c=25	550
5	6 (swb db)	-c=0	4200
6	2 (hammer)	-w=16384	220
7	2 (hammer)	-w=2048	1100
8	6 (swb db)	-w=2048	4700
9	2 (hammer)	-w=256	5300
10	6 (swb db)	-w=256	4400

La courbe ci-dessous (figure 5) reprend l'évolution des transactions au cours des différentes étapes du test.

Figure 5 : Évolution du nombre de transactions



On constate, grâce au tableau précédent et à la figure 5, que l'impact des fonctions de seuil supérieur et de pondération sur les performances est réel. Nous avons volontairement fortement modifié l'allocation des ressources sur le système afin de « forcer le trait » et d'avoir des résultats probants. Une gestion affinée de ces paramètres serait plus réaliste dans une logique d'exploitation d'environnements de production.

Conclusion

Force est de reconnaître que les outils d'ordonnancement disponibles avec Xen ont un réel impact sur la capacité à allouer les ressources CPU aux domaines prioritaires. Bien entendu, cette fonctionnalité n'a d'intérêt que sur des serveurs souffrant de contention.

Notons aussi que, dans une problématique d'équilibrage de charge et d'atteinte de SLO, Xen offre la possibilité de migrer une VM d'un serveur physique vers un autre serveur physique sans interruption de service (ou presque). Cette caractéristique, appelée « Vmotion » chez VMware, est une alternative, à l'équilibrage de charge au sein d'un même serveur hôte. Il est aussi indispensable de rappeler que l'augmentation ou la diminution des priorités n'a pas un effet linéaire sur la performance transactionnelle d'une application. C'est le principe du goulet d'étranglement (*bootleneck*), bien connu des testeurs de performance. Ceci pour souligner que les résultats affichés dans cet article ne peuvent en aucun cas servir de comparateur de performance avec d'autres serveurs, OS ou solutions de virtualisation.

Par ailleurs, nous n'avons pas eu la possibilité de mesurer la surcharge (*overhead*) générée par les processus Xen. Celle-ci est annoncée comme faible.

Enfin, dans un souci de professionnalisation, il serait souhaitable d'associer les options de pondération de l'ordonnanceur à un outil de monitoring des performances afin de lever une alerte en cas de non-atteinte d'un SLO puis, dans un deuxième temps, de modifier le poids d'une VM afin de lui octroyer plus ou moins de temps CPU. Ceci offrirait l'avantage d'adapter automatiquement les ressources aux contraintes business sans intervention humaine.

Yann Allandit,

HP, Sophia-Antipolis,
Yann.allandit@hp.com

RÉRÉRENCES

- [1] http://www.cl.cam.ac.uk/Research/SRG/netos/xen/
- [2] http://www.xensource.com/
- [3] http://virt-manager.et.redhat.com/download.html
- [4] http://hammerora.sourceforge.net/
- [5] http://www.dominicgiles.com/swingbench.html
- [6] http://www.oracle.com/technology/tech/oci/index.html
- [7] http://h20229.www2.hp.com/products/gplus/index.html
- [8] http://xosview.sourceforge.net/

GLOSSAIRE

ms : milliseconde.
SLO : *Service Level Objective* ou Objectif de niveau de service.
Host : Serveur physique exécutant Xen et hébergeant les VM.
VM : Machine virtuelle, invitée, domaine.
VCPU : CPU virtuelle (une ou plus par VM).
CPU : CPU physique disponible sur le système hébergeur.
Tick : Unité de click d'horloge (10 ms).
Time-slice : Temps octroyé à une VCPU avant d'être remplacée par une autre (30 ms).
Poids : Partage proportionnel de ressources CPU par les VM.
Cap : Seuil supérieur optionnel du temps CPU consommable par une VM.

Philippe « BooK » Bruhat

Perles de Mongueurs (Analyse de Logs)

Depuis le numéro 59, les Mongueurs de Perl vous proposent tous les mois de découvrir les scripts jetables qu'ils ont pu coder ou découvrir dans leur utilisation quotidienne de Perl. Bref, des choses trop courtes pour en faire un article, mais suffisamment intéressantes pour mériter d'être publiées. Ce sont les perles de Mongueurs.

Inspecter /var/log/mail.log avec Parse::Syslog::Mail

Tout est de la faute du spam

<mavie>

Mon fournisseur d'accès à Internet a décidé récemment de mettre en place l'authentification SMTP, afin de réduire le *spam* et la propagation des virus. L'idée est que les virus ne connaîtront pas l'utilisateur et le mot de passe et que leurs messages seront donc refusés (je pense qu'il suffit d'attendre une ou deux générations de virus pour voir apparaître des virus qui utiliseront l'authentification configurée dans Outlook, et tous ces efforts n'auront servi à rien).

Hélas, je n'ai jamais reçu d'email ou d'annonce d'aucune sorte de leur part pour me prévenir des modifications. C'est donc quand j'ai commencé à voir le contenu de ma *mail queue* se remplir avec des messages comme le suivant que je me suis douté de quelque chose.

```
$ mailq
-Queue ID- --Size-- ----Arrival Time---- -Sender/Recipient-------
207AA27FF52      636 Sun Jan  7 14:40:40  philippe.bruhat@free.fr
   (connect to smtp.tele2.fr[212.247.156.12]: Connection timed out)
                                        book@mongueurs.net

-- 0 Kbytes in 1 Request.
```

Après avoir changé temporairement de relais SMTP pour pouvoir communiquer avec le support, j'ai reçu l'explication suivante :

« Nous accusons réception de votre demande concernant l'utilisation de votre logiciel de messagerie et vous informons que depuis le 3 janvier, nous avons modifié le numéro de port sortant des mails.

Devant les nombreux *spams* envoyés sur les adresses mails et les virus, nous avons sécurisé l'envoi des e-mails en mettant en place la dépose authentifiée des emails.

Dorénavant, le numéro de port sortant est le 587 au lieu du 25.

Nous vous invitons à faire cette modification sur votre logiciel Postfix, pour tous les comptes mails utilisés. »

Après un bref haussement d'épaules (changer le port, c'est une mesure de sécurité ?), j'ai reconfiguré Postfix. Le mail partait enfin. Je croyais mon problème résolu.

Au bout de quelques jours, j'ai constaté que si les mails partaient bien, ils ne semblaient pas atteindre leur destination... Un bref coup d'œil au fichier `/var/log/mail.log` a confirmé mes soupçons :

```
    Jan  7 05:43:33 powie postfix/pickup[12584]: 601B727E444: uid=1000
from=<book>
    Jan  7 05:43:33 powie postfix/cleanup[14435]: 601B727E444:
message-id=<20070107044333.GA14057@localhost.localdomain>
    Jan  7 05:43:33 powie postfix/qmgr[27593]: 601B727E444:
from=<philippe.bruhat@free.fr>, size=2138, nrcpt=1 (queue active)
    Jan  7 05:43:38 powie postfix/smtp[14437]: 601B727E444:
to=<articles@mongueurs.net>, relay=smtp.tele2.fr[212.247.156.12]:587,
delay=4.9, delays=0.06/0.03/4.7/0.09, dsn=5.0.0, status=bounced (host
smtp.tele2.fr[212.247.156.12] said: 530 philippe.bruhat@free.fr There
is an error in your configuration. More info at www.tele2mail.com (in
reply to MAIL FROM command))
    Jan  7 05:43:38 powie postfix/bounce[14452]: 601B727E444: sender
non-delivery notification: 7411627E429
    Jan  7 05:43:38 powie postfix/qmgr[27593]: 601B727E444: removed
```

Le message `530 philippe.bruhat@free.fr There is an error in your configuration. More info at www.tele2mail.com (in reply to MAIL FROM command)` est assez inquiétant. En me connectant à la page web en question, j'ai donc découvert qu'il me fallait, en plus du changement de port, utiliser le nom d'utilisateur et le mot de passe de mon webmail chez eux (que je n'utilise jamais).

Avec l'aide de http://www.postfix.org/SASL_README.html#client_sasl, j'ai pu reconfigurer mon Postfix pour l'envoi de mails. Normalement, tout remarche correctement.

</mavie>

Et /var/log/mail.log, c'est le foutoir!

Le plus étonnant dans cette affaire, c'est que je n'ai reçu aucun *bounce*... Jusqu'à ce que je regarde plus attentivement les lignes de log qui suivent le rejet d'un message :

```
    Jan  7 05:43:38 powie postfix/cleanup[14435]: 7411627E429:
message-id=<20070107044338.7411627E429@powie.home.bruhat.net>
    Jan  7 05:43:38 powie postfix/qmgr[27593]: 7411627E429: from=<>,
size=4234, nrcpt=1 (queue active)
    Jan  7 05:43:38 powie postfix/bounce[14452]: 601B727E444: sender
non-delivery notification: 7411627E429
    Jan  7 05:43:43 powie postfix/smtp[14437]: 7411627E429:
to=<philippe.bruhat@free.fr>, relay=smtp.tele2.
fr[212.247.156.12]:587, delay=4.7, delays=0/0/4.6/0.09, dsn=5.0.0,
status=bounced (host smtp.tele2.fr[212.247.156.12] said: 530 <>
There is an error in your configuration. More info at www.tele2mail.
com (in reply to MAIL FROM command))
    Jan  7 05:43:43 powie postfix/qmgr[27593]: 7411627E429: removed
```

Les bounces étaient renvoyés à mon adresse d'expéditeur, en utilisant le même relais qui a refusé le message original. Les mêmes causes produisant les mêmes effets, les bounces ont donc fini avec mes autres envois, au fin fond de `/dev/null`.

Tous les mails que j'ai envoyés avec la mauvaise configuration de Postfix ont donc été perdus. Heureusement, comme je garde toujours une copie en `Fcc:` (*Folder Carbon-copy*) des messages envoyés, je peux les retrouver pour les renvoyer. Il suffit de retrouver les `Message-Id` des messages perdus dans le `/var/log/mail.log`.

Et c'est maintenant que nous allons enfin utiliser Perl... :-)

Comme nous l'avons vu ci-dessus, les logs de Postfix sont éclatés sur plusieurs lignes, en fonction du sous-système qui manipule le message. Pour faire mes recherches, j'aimerais plutôt disposer d'une structure de données simple qui contient toute l'information concernant un message.

C'est ce que permet de faire le module `Parse::Syslog::Mail`, créé par un de mes camarades mongueurs, Sébastien Aperghis-Tramoni. Ce module lit les logs *syslog* et en extrait les informations produites par les différents serveurs de messagerie. Pour des raisons de performance, `Parse::Syslog::Mail` ne fait aucune agrégation des logs qui correspondent au même message. Il se contente juste d'éclater chaque ligne de log dans un HASHREF en mettant à jour le plus d'informations possible.

Le script suivant réalise l'agrégation des informations correspondant au même message, à l'aide du champ `id` (qui est différent du champ `Message-ID` de SMTP). Ceci va permettre d'agréger toutes les informations (on a vu dans les extraits de log ci-dessus que des informations comme le `to` et le `from` ne sont pas sur les mêmes lignes de log). Attention cependant, des champs comme `text` ou `status` existent et sont différents pour chaque sous-système de Postfix ; pour un même message, les plus récents écraseront donc les plus anciens.

Cette opération d'agrégation est coûteuse en mémoire, car on ne peut pas savoir avec certitude quand on a vu tous les logs concernant un message en particulier, et on ne peut donc définir de critère pour le supprimer de la structure d'agrégation. Ma semaine de logs contenant un peu moins de 8000 messages, c'est tout à fait acceptable.

```
#!/usr/bin/perl
use strict;
use warnings;
use Parse::Syslog::Mail;  # nécessite la version 0.10
my $maillog = Parse::Syslog::Mail->new('-');
    # logs reçues sur STDIN
my $message = {};
    # logs agrégées
my $lost    = {};
    # messages perdus
while ( my $log = $maillog->next() ) {
    # agrégation des logs
    $message->{ $log->{id} }{$_} = $log->{$_} for keys %$log;

    # recherche des messages perdus
    if ( $log->{status} && $log->{status} =~ /^bounced: / ) {
        my $msg = $message->{ $log->{id} };
        next if $msg->{from} eq '<>';
          # ignore les bounces des bounces

        # ajoute les destinataires au message
        $lost->{ $msg->{msgid} } .=
          sprintf "    from: %s to: %s\n", @{$msg}{qw( from to )};
    }
}

# affiche la liste des messages perdus
print "$_:\n$lost->{$_}" for keys %$lost;
```

Les messages perdus sont détectés grâce au statut `bounced`. Les messages du *mailer-daemon* étaient envoyés avec un `from` à `<>`, et sont donc ignorés.

Une agrégation supplémentaire est réalisée cette fois sur le champ `Message-ID` (ou `msgid` pour `Parse::Syslog::Mail`) afin de n'avoir qu'un bloc d'information par message effectivement envoyé.

En tant que `root`, on crée un flux avec tous les logs et on l'envoie sur notre script :

```
# cd /var/log
# ( zcat mail.log.*.gz ; cat mail.log.0 mail.log ) | ./bounced
<20070107044333.GA14057@localhost.localdomain>:
    from: <philippe.bruhat@free.fr> to: <articles@mongueurs.net>
<20070106001958.GA27004@localhost.localdomain>:
    from: <philippe.bruhat@free.fr> to: <brian.d.foy@gmail.com>
<20070106184330.GA2163@localhost.localdomain>:
    from: <philippe.bruhat@free.fr> to: <articles@mongueurs.net>
<20070107122844.GA14546@localhost.localdomain>:
    from: <philippe.bruhat@free.fr> to: <wendy@dijkmat.nl>
    from: <philippe.bruhat@free.fr> to: <ann@domaintje.com>
<20070107122505.GA12081@localhost.localdomain>:
    from: <philippe.bruhat@free.fr> to: <board@yapceurope.org>
```

J'ai pu ainsi retrouver les 15 messages perdus pendant ces deux jours, et les renvoyer à leurs destinataires. Ouf !

À vous !

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

Philippe « BooK » Bruhat

book@mongueurs.net,
de Lyon.pm et Paris.pm.

J.-M. Friedt

Introduction à la programmation sur PlayStation Portable

Nous proposons de présenter les outils de développement libres pour la console de jeux PlayStation Portable. Après l'installation de la chaîne de cross-compilation et un premier exemple simple d'affichage graphique, nous interfacerons un récepteur GPS à la console et afficherons les coordonnées GPS courantes ou le trajet parcouru. L'objectif est de géolocaliser les points d'accès wifi détectés au moyen de l'interface de communication sans fil dont est équipée la console et de réaliser ainsi un outil de wardriving peu encombrant et présentant une autonomie de quelques heures.

1 Introduction

Le développement sur console de jeux portable est un moyen pratique de se familiariser avec la plupart des concepts du développement logiciel pour systèmes embarqués tout en profitant d'une plate-forme de développement peu coûteuse (par rapport aux systèmes industriels dédiés), avec de nombreuses interfaces fournies par défaut (touches, écran, périphériques de communication) et une puissance de calcul digne des ordinateurs actuels **[1, 2]**. Cependant, les principaux fabricants de consoles de jeux n'ont jamais été favorables au développement amateur de logiciels pour leur matériel et le programmeur désireux d'appliquer ses compétences à ces plateformes a dû tant bien que mal trouver les ressources nécessaires.

Nous allons ici présenter les outils de développement disponibles pour la PlayStation Portable (PSP) de Sony : cette console fournit un écran couleur de résolution acceptable (480×272 pixels), un processeur central basé sur une architecture MIPS[1] cadencé à 333 MHz **[3]**, de nombreux périphériques (ports USB, série asynchrone, wifi) et un support de stockage de masse peu coûteux et accessible depuis un PC fonctionnant sous GNU/Linux (le *Sony Memory Stick*). Nous allons tenter d'utiliser au mieux ce matériel dans un objectif d'acquisition et de visualisation de données. Il ne sera pas ici question de développement de jeux qui nécessite des méthodes de programmation très particulières que l'auteur ne maîtrise pas.

Nous supposerons dans la suite de ce document que nous disposons d'un Sony Memory Stick et d'un adaptateur permettant de connecter une telle carte mémoire sur un PC, et éventuellement d'un câble de liaison PSP-USB.

[1] Architecture modulaire très utilisée dans les systèmes embarqués tels que les points d'accès Broadcom : http://www.linux-mips.org/wiki/Systems.

2 Installation des outils de développement

Plusieurs langages permettant le développement d'applications pour la PSP (dits « *homebrew* » dans la communauté de développeurs PSP) sont disponibles : mentionnons ici notamment les langages interprétés tels que Lua ou Python. Nous nous intéresserons ici cependant directement à la solution la plus efficace, à savoir l'utilisation de gcc pour compiler du code C ou assembleur à destination du processeur MIPS de la PSP.

2.1 Mise à jour du firmware de la console

La première étape pour rendre une PlayStation Portable (PSP) utilisable par un développeur est de revenir à un *firmware* permettant l'exécution depuis le support de stockage de masse Memory Stick (MS). En effet, historiquement, Sony n'avait pas pris soin d'empêcher l'exécution d'un programme depuis ce support et cette fonctionnalité inattendue sembla poser un problème de sécurité. Il s'en est suivi une séquence de mises à jour du firmware par Sony interdisant l'exécution de programmes depuis le MS, et la mise à disposition par les développeurs indépendants de méthodes pour ramener sa PSP à un firmware acceptant de telles fonctionnalités. Le firmware de prédilection pour le développeur a pour version 1.5. Notre premier objectif est donc de reflasher le firmware de la PSP de sa version disponible à l'achat vers la version 1.5 qui autorise l'exécution d'un programme depuis le support de masse MS **[4]**.

ATTENTION

La PSP deviendra inutilisable si cette opération échoue. Il est fondamental de bien se documenter sur les dangers de flasher sa PSP afin d'en remplacer le firmware courant par celui de la version 1.5. L'auteur ne saurait être tenu responsable pour tout dommage résultant d'un échec de cette mise à jour.

La version du firmware sur une PSP achetée en France en décembre 2006 est 2.01. Le passage du firmware 2.01 au firmware 1.5 est bien documenté à http://gueux-forum.net/index.php?showtopic=125133. Un MS d'au moins 32 MB est pour cela nécessaire. Cette mise à jour efface bien entendu tous les logiciels ajoutés par Sony entre la version 1.5 et la version fournie au moment de l'achat de sa console – notamment le logiciel d'affichage de pages web. Ces fonctionnalités peuvent être retrouvées en repassant provisoirement sur un firmware plus récent grâce à Devhook. Ce logiciel permet en effet, depuis un firmware 1.50, de charger en mémoire volatile un firmware plus récent et les applications associées, et ce, jusqu'à la réinitialisation de la console (par sa mise hors tension par exemple). Nous avons utilisé à ces fins la version de Devhook

incluant toutes les mises à jour, version 0.44 disponible à http://3mul.free.fr/vl/module.php?page=programme_detail&type_programme=emulateurs&i=63 qui a le bon goût de déjà inclure tous les firmwares à utiliser. Il n'est pas clair que les versions plus récentes que 1.50 des firmwares amènent des améliorations notables, si ce n'est de brider l'utilisateur dans les applications possibles de sa PSP : l'accès aux firmwares plus récents ne semble donc pas d'un intérêt remarquable, si ce n'est de se rassurer sur la disponibilité des quelques logiciels ajoutés récemment par Sony.

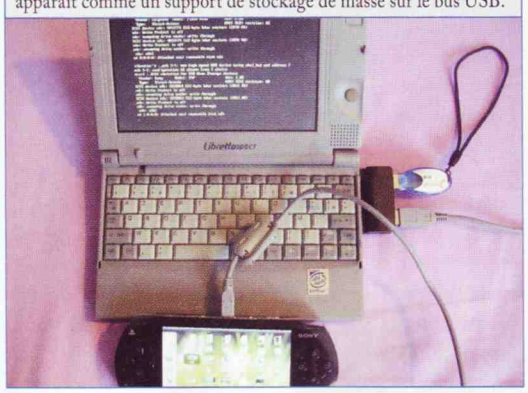
La phase la plus dangereuse est passée : nous avons maintenant une PSP capable d'exécuter un logiciel depuis le support MS. Nous allons maintenant aborder les aspects d'installation des outils de compilation de nos programmes sous GNU/Linux à destination du processeur MIPS de la PSP (*cross-compilation*).

2.2 Installation de la chaîne de cross-compilation

La cross-compilation de code à destination du processeur MIPS de la PSP depuis l'ordinateur sur lequel tourne GNU/Linux (ici un processeur compatible Intel x86) nécessite la compilation du trio habituel gcc (version 4.0.2), binutils (version 2.16.1) et newlib (version 1.13.0). Les développeurs pour PSP ont considérablement simplifié le travail en fournissant un script, `toolchain.sh` dans l'archive `psptoolchain-20060120.tgz` disponible à http://ps2dev.org/psp/Tools, qui se charge de récupérer les bonnes versions des codes sources, leur appliquer les patchs nécessaires pour supporter l'architecture PSP, et de les compiler avec les bonnes options – principalement `-target=psp`. Une fois psp-gcc installé dans son emplacement par défaut `/usr/local/pspdev/bin`, il ne nous reste plus qu'à compiler le PSPSDK (obtenu par CVS auprès de svn://svn.pspdev.org/psp/trunk/pspsdk) afin d'obtenir tous les outils nécessaires à la génération de binaires prêts à être exécutés depuis un firmware 1.5 sur la PSP. Cette opération est assez lourde puisqu'elle nécessite entre 32 et 64 MB de RAM et environ 1,5 GB d'espace disque.

L'accès à la console de jeu se fera soit après installation d'iRShell disponible à http://dl.qj.net/index.php?pg=12&fid=11531&catid=190 qui permet d'accéder à la PSP comme à une clé de stockage de masse sur port USB (Fig. 1), soit via le MS.

Figure 1 : Connexion de la PSP à un ordinateur portable sous Debian GNU/Linux sur lequel est installé le PSP SDK : la console apparaît comme un support de stockage de masse sur le bus USB.



3 Premier exemple : la fractale de Mandelbrot

De nombreux tutoriaux sont disponibles sur le web, et ceux qui semblent les plus pertinents sont disponibles à http://www.psp-programming.com/tutorials/. En nous inspirant de ces présentations, nous allons commencer par développer une petite application simple affichant sur l'écran de la PSP la fractale de Mandelbrot (accès au mode graphique) et sauvant sur MS le résultat du calcul pour exploitation ultérieure.

La fractale de Mandelbrot **[5]** est définie comme l'ensemble des points `c` dans le plan complexe pour lesquels l'itération de la suite $z_{n+1}=z_n^2+c$ $(z_0=c)$ ne diverge pas. Nous affichons pour chaque point `c` le nombre d'itérations, borné à un maximum de 16, au bout duquel la suite a été identifiée comme divergente (c.-à-d. la couleur du point `c` est `n` lorsque $|z_n|>2$, pour $n<17$ – l'ensemble des points blancs sur la Fig. 2 présente donc les valeurs de `c` pour lesquelles la suite n'a pas divergé au bout de 16 itérations).

```
// 29/12/2006
// http://psp.jim.sh/pspsdk-doc/
#include <pspkernel.h>
#include <pspctrl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "graphics.h"

PSP_MODULE_INFO ("mandel JMF", 0x1000, 1, 1);
PSP_MAIN_THREAD_ATTR (0);

#define xmin -2.0
#define xmax 1.0
#define ymin -1.5
#define ymax 1.5

void
nextname (char *base)
{
  char s[52];
  int fcpt = 0;
  SceUID f;
  do
    {
      sprintf (s, "ms0:/%s%d.dat", base, fcpt);
      f = sceIoOpen (s, PSP_O_RDONLY, 0777);
      if (f > 0)
 {
   sceIoClose (f);
   fcpt++;
 } // f>=3 if file exists
    }
  while (f > 0);
  sprintf (base, "%s", s);
}
int
main (void)
{
  SceCtrlData pad;
   int cpt = 0, i, ix, iy;
  double x, y, xt, yt, tmp;
```

```
SceUID f;
 char s[42];

 // ajout anti-freeze (cf web)
 SetupCallbacks ();
 //  2 bytes/pixel && 512 instead of 480
 InitializeGraphics (1);
//  blank screen
//  memset((char*)0x44000000,0,512*272*2);
  sprintf (s, "mandel");
  nextname (s);
  f = sceIoOpen (s, PSP_O_WRONLY | PSP_O_CREAT, 0777);
  for (iy = 0; iy < 272; iy++)
    for (ix = 0; ix < 480; ix++)
      {
cpt = 0;
x = (xmax - xmin) / 480. * (double) ix + xmin;
y = (ymax - ymin) / 272. * (double) iy + ymin;
xt = x;
yt = y;
do
  {
    tmp = xt * xt - yt * yt + x;
    yt = 2. * xt * yt + y;
    xt = tmp;
    cpt++;
  }   // z -> z^2+c
      // R=sup(|c|,2)
while (((xt * xt + yt * yt) < 4) && (cpt < 16));
if (cpt == 0)
  cpt = 1;
PlotPixel (ix, iy, cpt * 16 - 1, cpt * 16 - 1,
    cpt * 16 - 1);
sceIoWrite (f, &cpt, 1);
      }
  PlotPixel (1, 1, 255, 255, 255);
  sceIoClose (f);
  do
  {
     sceCtrlReadBufferPositive (&pad, 1);
     sceDisplayWaitVblankStart ();
   }
  while (!(pad.Buttons & PSP_CTRL_CROSS));
  sceKernelExitGame ();
  return 0;
}

//////////////////////////////////////////////////////////
// http://www.psp-programming.com/tutorials/c/lesson02.htm
//////////////////////////////////////////////////////////
/* Exit callback */

   int
exit_callback (int arg1, int arg2, void *common)
{
  sceKernelExitGame ();
  return 0;
}

/* Callback thread */
int
CallbackThread (SceSize args, void *argp)

{
  int cbid;

  cbid =
   sceKernelCreateCallback ("Exit Callback", exit_callback,
       NULL);
  sceKernelRegisterExitCallback (cbid);
  sceKernelSleepThreadCB ();
  return 0;
}

/* Sets up the callback thread and returns its thread id */
int
SetupCallbacks (void)
{
  int thid = 0;

  thid =
    sceKernelCreateThread ("update_thread",
CallbackThread,
      0x11, 0xFA0, 0, 0);
  if (thid >= 0)
    {
      sceKernelStartThread (thid, 0, 0);
    }
  return thid;
}
```

Tableau 1 : Exemple de programme affichant la fractale de Mandelbrot et stockant sur MS les résultats du calcul. Aucune optimisation n'a été incluse dans ce code – qu'il s'agisse de l'affichage ou de l'algorithme de calcul – afin de le garder aussi lisible que possible : la durée d'exécution du calcul est de 37 secondes.

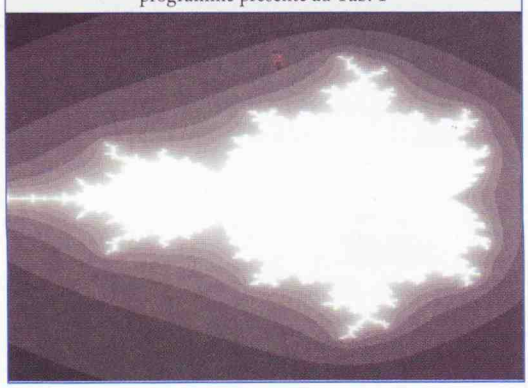
Le Makefile associé au programme (Tab. 1) est présenté dans le Tab. 2 : il s'appelle, pour générer un exécutable pour firmware 1.5, par la commande `make kxploit` qui génère deux sous-répertoires, `mandel` et `mandel%` dans l'exemple qui nous intéresse ici (le nom de l'exécutable présenté par la PSP et des sous-répertoires associés sont définis par la variable `PSP_EBOOT_TITLE` du Makefile). Ces deux répertoires sont placés sur le MS dans le sous répertoire `PSP/GAME` afin d'être reconnus lors de leur exécution sur la PSP. Si plusieurs projets sont placés dans le même sous-répertoire, on prendra soin, avant chaque nouvelle compilation, d'effacer les objets et le fichier `PARAM.SFO` qui contient les paramètres de génération de l'exécutable `EBOOT.PBP` inclus dans ces répertoires.

```
# make -f simple_rs232.Makefile kxploit
TARGET = mandel
OBJS = mandel.o
USE_PSPSDK_LIBC = 1
INCDIR =
CFLAGS = -O2 -G0 -Wall
CXXFLAGS = $(CFLAGS) -fno-exceptions -fno-rtti
ASFLAGS = $(CFLAGS)
LIBDIR =
LDFLAGS =
EXTRA_TARGETS = EBOOT.PBP
PSP_EBOOT_TITLE = mandel_JMF
PSPSDK=$(shell psp-config --pspsdk-path)
include $(PSPSDK)/lib/build.mak
LIBS += -lpsphprm_driver -lpspkernel -lpsppower -lm -lpng -lz -lc -lpspkernel
# -lc *doit* etre avant -lpspkernel
```

Tableau 2 : Fichier de configuration Makefile associé au programme présenté au Tab. 1.

Nous utilisons pour l'accès au mode graphique une bibliothèque `graphics.h` disponible dans les exemples de http://www.freewebs.com/jason867/psphomebrew.htm. Le principal intérêt de ce code réside dans la mise en évidence de l'adresse de la mémoire vidéo et la procédure à suivre pour y écrire (fonction `PlotPixel()`) ou en lire le contenu pour effectuer une capture d'écran.

Figure 2 : Capture d'écran du résultat de l'exécution du programme présenté au Tab. 1



Nous constatons dans cet exemple que l'appel aux fonctions du système d'exploitation exécuté par la PSP implique un certain nombre de changements de nomenclature par rapport à l'ANSI C classique. Par exemple, l'ouverture de fichier habituelle par `fopen()` est remplacée par `sceIoOpen()` avec les mêmes arguments, et `fclose()` est remplacé par `sceIoClose()`. Le reste du code suit une syntaxe tout à fait classique par ailleurs.

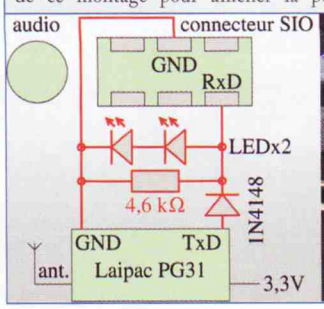
Une autre possibilité pour l'affichage en mode texte d'informations sur la PSP est l'utilisation du mode Debug : initialisé par la procédure `pspDebugScreenInit();`, il permet d'afficher des caractères ASCII au moyen de la fonction `pspDebugScreenPrintf()` qui accepte une liste d'arguments identique à celle dont nous avons l'habitude avec `printf()`. Nous utiliserons plus bas ce mode pour afficher les trames NMEA issues d'un récepteur GPS connecté à la PSP.

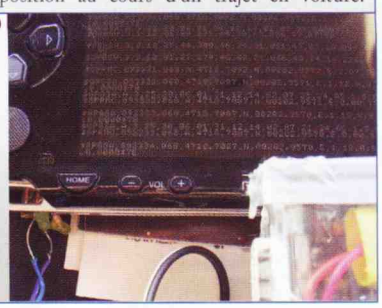
4 Communication par le port série asynchrone

Nous allons dépasser le cadre des tutoriaux sus-cités pour aborder les aspects d'accès aux ports de communication de la PSP, aspect le plus intéressant pour faire interagir la console avec le monde qui l'entoure.

Nous avons présenté à plusieurs reprises dans ces pages **[6,7]** des développements autour de récepteurs GPS OEM de coût raisonnable au vu de leurs performances. Cependant, un point délicat avec l'enregistrement en aveugle de trames GPS au cours d'un trajet est l'incapacité de vérifier en temps réel le bon fonctionnement du logiciel d'acquisition : rien n'est plus frustrant que de rentrer d'une longue randonnée et de se rendre compte qu'un élément du circuit d'acquisition n'a pas fonctionné correctement. Nous allons ici utiliser la PSP comme écran d'affichage des trames acquises, comme support de stockage de masse non volatile de secours pour conserver les traces du trajet, pour finalement combiner ces fonctionnalités avec l'acquisition géolocalisée de quantités physiques (dans le cas qui nous intéressera plus bas, la puissance radiofréquence reçue depuis des points d'accès wifi).

Figure 3 : Gauche : montage connectant un GPS à une PSP. Les composants passifs ont pour vocation de protéger la PSP contre tout signal indésirable (la diode 1N4148 élimine tout risque d'appliquer un potentiel négatif sur la broche RxD, tandis que les deux LED en série limitent le potentiel appliqué à RxD à environ 3,4 V. La résistance en parallèle avec les deux LED permet alors au potentiel de RxD de redescendre lors du passage d'un bit 1 à un bit 0). Nous avons pu connecter un récepteur GPS Laipac PG31 (alimenté en 3,3 V et communiquant donc sur un signal haut à cette tension) sans composant passif de protection. Noter qu'un MAX(3)232 comporte non seulement une pompe de charge générant la tension négative nécessaire au protocole RS232, mais aussi des inverseurs : il ne suffit donc pas de connecter une diode pour faire communiquer un PC avec la PSP, mais en plus un inverseur. Droite : utilisation de ce montage pour afficher la position au cours d'un trajet en voiture.





Le connecteur à 6 broches situé à côté de la prise de l'écouteur audio (en bas à gauche de la PSP) contient tous les signaux pour permettre une communication asynchrone compatible avec la norme RS232 **[8]**. Les niveaux des signaux sont entre 0 V et 2,5 V nominaux, compatibles avec un signal haut à 3,3 V. Nous allons connecter un récepteur GPS Laipac PG-31[2]. La connexion ne nécessite aucune précaution particulière bien que nous ayons choisi de placer deux LED afin de protéger le port série de la PSP de tout risque de surtension (une diode Zener ferait le même effet).

Une fiche capable de s'insérer dans l'emplacement prévu à côté du connecteur audio doit être réalisée : nous gravons pour ceci un circuit imprimé double-face (époxy FR4 de 1,6 mm d'épaisseur) avec 3 pistes de chaque côté, espacées de 1,5 mm et de 1 mm de largeur. Le lecteur n'ayant pas accès au matériel nécessaire pour graver un circuit imprimé pourra détruire une carte PCI pour en récupérer trois broches du connecteur prévu pour s'enficher dans une carte mère de PC (Fig. 5).

```
// 26/12/2006
// http://psp.jim.sh/pspsdk-doc/
#include <pspkernel.h>
#include <pspdebug.h>
#include <pspdisplay.h>
#include <pspsdk.h>
#include <pspctrl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <psppower.h>
#include <math.h>
#include <png.h>
#include "graphics.h"
```

[2] Disponible auprès de Lextronic, www.lextronic.fr, pour 83 euros TTC : http://www.lextronic.fr/laipac/tf30.htm.

```
// 1000 = set kernel mode
PSP_MODULE_INFO ("GPS JMF", 0x1000, 1, 1);
PSP_MAIN_THREAD_ATTR (0);

#define printf pspDebugScreenPrintf

void
nextname (char *base)
{
  char s[52];
  int fcpt = 0;
  SceUID f;
  do
    {
      sprintf (s, "ms0:/%s%d.dat", base, fcpt);
      f = sceIoOpen (s, PSP_O_RDONLY, 0777);
      if (f > 0)
 {
   sceIoClose (f);
   fcpt++;
 } // f>=3 if file exists
    }
  while (f > 0);
  sprintf (base, "%s", s);
}
int
main (void)
{
  SceCtrlData pad;
  int bauds[7] =
    { 1200, 2400, 4800, 9600, 19200, 38400, 57600 };
  int cpt = 2, ch, fcpt = 0, flag = 0, ilat, ilon, i;
  double flat, flon, oflat = 0., oflon = 0., mag =
    128., degla, deglo;
  char sauv = 0, s[52], lat[16], lon[16], txt = 1;
  SceUID f, scr;

  SetupCallbacks (); // ajout anti-freeze (cf web)
  pspDebugScreenInit (); // init text mode
  pspDebugScreenSetTextColor (0xffffffff); // White
  pspDebugScreenPrintf
    ("[] and O to change baud rate, ^ to toggle save\n");
  pspDebugScreenPrintf
    ("UP to toggle txt/graphic mode, X to validate and quit");
  pspDebugScreenSetXY (1, 6);
  pspDebugScreenPrintf ("Select: clear screen\n");
  pspDebugScreenPrintf ("Start: dump screen\n");
  pspDebugSioInit (); // init RS232

  while (1)
    {
      pspDebugScreenSetXY (1, 3);
      pspDebugScreenPrintf ("baud=%d  \tsave=%d \ttxt=%d ",
       bauds[cpt], sauv, txt);
      sceCtrlPeekBufferPositive (&pad, 1);
      if (pad.Buttons & PSP_CTRL_SQUARE)

 {
   cpt++;
   if (cpt > 6)
     cpt = 0;
 }
      if (pad.Buttons & PSP_CTRL_CIRCLE)
 {
   cpt--;
   if (cpt < 0)
     cpt = 6;
 }
      if (pad.Buttons & PSP_CTRL_TRIANGLE)
 {
   sauv = 1 - sauv;
 }
      if (pad.Buttons & PSP_CTRL_UP)
 {
  txt = 1 - txt;
 }
      if (pad.Buttons & PSP_CTRL_CROSS)
 {
   break;
 }
      // delay 200 ms = debounce
      sceKernelDelayThread (200000);
    }
  printf ("\n");
/*
// write dummy file
  f=sceIoOpen("ms0:/gps0.dat",PSP_O_WRONLY|PSP_O_CREAT,0777);
  for (i=32;i<127;i++) {sceIoWrite(f,&i,1);printf("%c",i);}
  sceIoClose(f); printf("\n");
// read dummy file
  ch=0;
  f=sceIoOpen("ms0:/gps0.dat",PSP_O_RDONLY,0777);
  do {res=sceIoRead(f,&i,1);printf("%c",i);ch++;}
  while ((res!=0)&&(ch<100));
  sceIoClose(f); printf("\n");
*/
// let's start
  if (sauv == 1)
    {
      sprintf (s, "gps");
      nextname (s); // returns in s the new filename
      f =
 sceIoOpen (s,
     PSP_O_WRONLY | PSP_O_CREAT |
     PSP_O_APPEND, 0777);
    }
  pspDebugSioSetBaud (bauds[cpt]);
  printf ("\nPSP GPS JMF test");
  if (sauv == 1)
    pspDebugScreenPrintf (": saving in %s\n", s);
  else
    pspDebugScreenPrintf ("\n");
  sceKernelDelayThread (1000000); // delay 1 s
  if (txt == 0)
    {
      //  2 bytes/pixel && 512 instead of 480
      InitializeGraphics (1);
      // blank screen
      memset ((char *) 0x44000000, 0, 512 * 272 * 2);
    }
  cpt = 0;
  fcpt = 0;
  while (1)
    {

      // get new RS232 char
      ch = pspDebugSioGetchar ();
      if ((ch >= 0) && (ch != '\r'))
 {
   if (sauv == 1)
     {
       sceIoWrite (f, &ch, 1);
     }   // save in File
   if (txt == 1)
     {
       pspDebugScreenPrintf ("%c", ch);
     }   // print
   if (((ch == 'G') || (ch == 'P') || (ch == 'A')
```

Disponible chez votre
marchand de journaux
et sur
http://www.ed-diamond.com

Apprivoisez votre pingouin !

GNU LINUX PRATIQUE

En KIOSQUE

N°40

KNOPPIX 5.1.1 BUREAU 3D AIGLX...

ENFIN UNE SOLUTION DE LECTURE / ÉCRITURE SUR DISQUES XP ET VISTA !

Cahier We

découvrir 12/21
- SAUVEGARDEZ AVEC KEEP : VOS DONNEES ENFIN EN SECURITE !
- @LEX POLL 2.1 : LES SONDAGES FACILES POUR VOS SITES

écouter/voir 26/33
- LA MAÎTRISE DES SOUS-TITRES AVEC GNOME SUBTITLES
- TRAVAILLEZ LES TRANSITIONS AUDIO DANS KINO

communiquer 36
L'IRC, LA MESSAGERIE INSTANTANEE PUISSANCE DIX !

configurer 46
NOUVELLES DISTRIBUTIONS : FAUT-IL INSTALLER OU METTRE À JOUR ?

sur le CD-ROM
Knoppix 5.1.1
LA DISTRIBUTION LIVE LA PLUS ÉVOLUÉE À CE JOUR, INCLUANT UN BUREAU 3D KDE 3.5.5 + BERYL, FIREFOX 2, OPENOFFICE.ORG 2.1, X.ORG 7.1 AIGLX ET DES FONCTIONNALITÉS AVANCÉES DE LECTURE/ÉCRITURE DES DISQUES WINDOWS XP (NTFS AVEC NTFS-3G)
UTILISEZ LINUX PARTOUT ET ACCÉDEZ SIMPLEMENT AUX DONNÉES WINDOWS DE LA MACHINE
voir p. 4 pour une description détaillée

DÉCOUVRIR :
60 LES BONS TUYAUX POUR BOOSTER LES VISITES SUR VOTRE BLOG
71 PHPMYVISITES, ENFIN UN OUTIL SIMPLE ET PUISSA POUR ANALYSER L'AUDIENC DE VOTRE SITE

COMPRENDRE :
74 MAÎTRISEZ LES PSEUDO FORMATS CSS2 :BEFORE ET :AFTER

S'ENTRAÎNER :
76 AJOUTEZ UNE BORDUR ENTRE LES ÉLÉMENTS D'U LISTE HTML
80 UTILISEZ OVERLIB POUR CRÉER DES INFOBULLES ÉVOLUÉES

FRANCE MÉTRO : 5,95 € - DOM 6,60 €
BEL - LUX - PORT. CONT. : 6,85 €
CH : 12 CHF - CAN : 11 $ C - MAR : 65 DH
L 18864 - 40 - F: 5,95 € - RD

```
      || (flag > 99)))
    flag++;
  else
    flag = 0;
  if (flag == 5)
    {
      flag = 100;
      fcpt = 0;
    }   // on a eu GPGGA
  if ((flag > 112) && (flag < 122))
    {
      lat[flag - 113] = ch;
    }
  if ((flag > 124) && (flag < 135))
    {
      lon[flag - 125] = ch;
    }
  if (flag == 135)
    {   // fini parsing $GPGGA
      lat[9] = 0;
      lon[10] = 0; // end of string
      flat = atof (lat);
      flon = atof (lon);
      degla = floor (flat / 100.) * 100.;
      deglo = floor (flon / 100.) * 100.;
            // 0..60 -> 0..100 lat
      flat = degla + (flat - degla) / 60. * 100.;
            // 0..60 -> 0..100 lon
      flon = deglo + (flon - deglo) / 60. * 100.;
      if ((oflat == 0.) && (oflon == 0.))
{
  oflat = flat;
  oflon = flon;
}  // init pos
      ilat =
(int) (floor ((flat - oflat) * mag)) + 136;
      ilon =
(int) (floor ((flon - oflon) * mag)) + 240;
      if (txt == 1)
{
  printf ("\n -> %lf/%lf %lf/%lf mag=%lf\n",
   flat, oflat, flon, oflon, mag);
}
      else
if ((ilat > 0) && (ilat < 272) && (ilon > 0)
    && (ilon < 480))
PlotPixel (ilon, 271 - ilat, 255, 255, 255);
      flag = 0;
    }
}
     sceCtrlPeekBufferPositive (&pad, 1);// DON'T wait VSync
     if (pad.Buttons & PSP_CTRL_CROSS)
{
  break;
}
     if (fcpt == 0)
{

  if (pad.Buttons & PSP_CTRL_LEFT)
    {
      oflon -= 10.;
      fcpt = 1;
    }
  if (pad.Buttons & PSP_CTRL_RIGHT)
    {
      oflon += 10.;
      fcpt = 1;
    }
```

```
  if (pad.Buttons & PSP_CTRL_UP)
    {
      oflat += 10.;
      fcpt = 1;
    }
  if (pad.Buttons & PSP_CTRL_DOWN)
    {
      oflat -= 10.;
      fcpt = 1;
    }
  if (pad.Buttons & PSP_CTRL_SQUARE)
    {
      mag *= 2.;
      fcpt = 1;
    }
  if (pad.Buttons & PSP_CTRL_CIRCLE)
    {
      if (mag > 1)
mag /= 2.;
      fcpt = 1;
    }
  if (pad.Buttons & PSP_CTRL_SELECT)
    {
            // blank screen
      memset ((char *) 0x44000000, 0, 512 * 272 * 2);
      oflat = 0.;

     oflon = 0.;
    }
  if (pad.Buttons & PSP_CTRL_START)
    {
      sprintf (s, "dump");
      nextname (s); // returns the new filename
      scr =
 sceIoOpen (s,
     PSP_O_WRONLY | PSP_O_CREAT |
     PSP_O_APPEND, 0777);
      sceIoWrite (scr, lat, 9); // dump screen
      sceIoWrite (scr, lon, 10);
      for (i = 0; i < 272; i++)
 {
   sceIoWrite (scr, (char *) (0x44000000 + 2 * i * 512), 480 * 2);
   PlotPixel (1, i, 155, 155, 155);
 }
      sceIoClose (scr);
      // screenshot("ms0:/d.png");
    }
}
  }
 if (sauv == 1)
   sceIoClose (f);
 if (txt == 1)
   pspDebugScreenPrintf ("Good bye");
 sceKernelExitGame ();
 return 0;
}
////////////////////////////////////////////////////////////
// http://www.psp-programming.com/tutorials/c/lesson02.htm
////////////////////////////////////////////////////////////
/* Exit callback */
int
exit_callback (int arg1, int arg2, void *common)
{
  sceKernelExitGame ();
  return 0;
}
/* Callback thread */
int
CallbackThread (SceSize args, void *argp)
```

```
{
  int cbid;
  cbid =
    sceKernelCreateCallback ("Exit Callback", exit_callback,
        NULL);
  sceKernelRegisterExitCallback (cbid);
  sceKernelSleepThreadCB ();
  return 0;
}
/* Sets up the callback thread and returns its thread id */
int
SetupCallbacks (void)
{
  int thid = 0;
  thid =
  sceKernelCreateThread ("update_thread", CallbackThread,
      0x11, 0xFA0, 0, 0);
  if (thid >= 0)
    {
      sceKernelStartThread (thid, 0, 0);
    }
  return thid;
}
```

Tableau 3 : Exemple de programme permettant de sélectionner la vitesse de communication du GPS sur son port RS232 (4800 bauds par défaut pour le Laipac PG-31), puis d'afficher les trames NMEA brutes issues du récepteur, ou de traiter ces trames pour en extraire latitude et longitude pour restituer les informations de façon graphique.

Un exemple de code minimaliste permettant de capturer les trames sur le port série et de les afficher est disponible à http://forums.ps2dev.org/viewtopic.php?t=6294. On y trouve l'initialisation de la vitesse de transfert par la commande `pspDebugSioSetBaud(38400);` que nous adaptons à nos propres besoins.

Nous observons cependant que ce code contient une boucle de taille prédéfinie qui ne saurait convenir pour une application générale. Nous allons donc remplacer cette boucle par une condition de boucle infinie (`while (1) { ...}`) et conditionner la sortie de cette boucle par l'appui d'un bouton sur la PSP. Cependant, l'utilisation de la fonction bloquante d'interrogation des touches `sceCtrlReadBufferPositive()` pour détecter l'appui d'une touche que nous associerions à l'arrêt du programme est trop lente et nous fait perdre des caractères. La solution qui nous a semblé appropriée est l'appel à la fonction non bloquante `sceCtrlPeekBufferPositive()` qui – n'étant pas bloquante jusqu'au prochain rafraîchissement de l'écran – nous permet d'interrompre proprement le programme (par appel à la fonction `sceKernelExitGame();`) sans perdre de caractère transmis par le port série. Nous profitons de la gestion de l'appui des touches pour proposer la translation de l'origine de la carte affichée à l'écran, l'effacement ou la capture d'écran (Fig. 4).

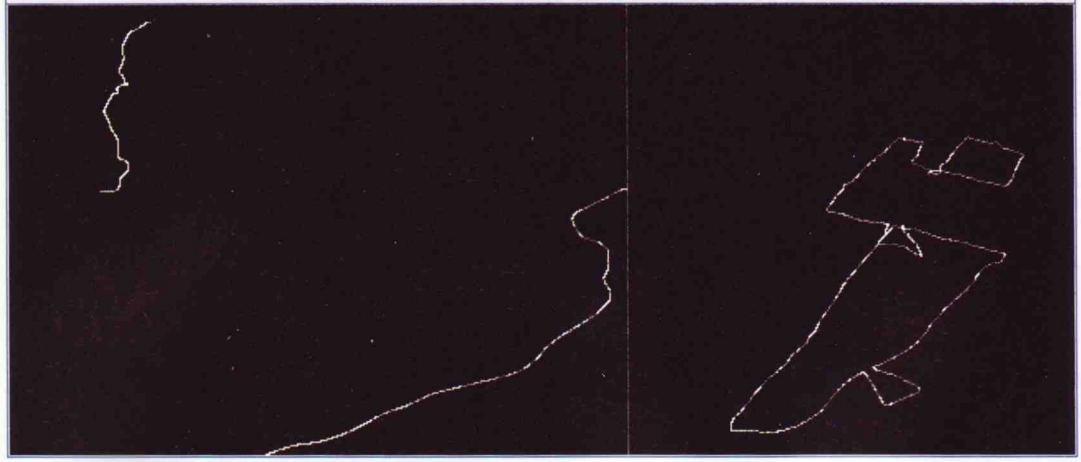
Il apparaît à l'usage que les fichiers créés pour les sauvegardes sur MS ne sont créés qu'au moment de leur fermeture. Cela signifie que les données accumulées ne sont réellement stockées que lors de la fermeture du fichier, c'est-à-dire à la fin d'une séance d'acquisition de données. Aucune solution satisfaisante n'a pour le moment été identifiée pour conserver les données en cours d'utilisation du programme, avant de quitter proprement et de fermer le fichier (`sceIoClose()` qui se comporte exactement comme l'habituel `fclose()`). Il ne semble pas exister de fonction pour vider les tampons (fonction `fflush()` classiquement) sur PSP. L'ensemble des fonctions spécifiques au PSP SDK sont décrites à http://psp.jim.sh/pspsdk-doc/.

Nous sommes donc capables, à ce stade d'avancement, de capturer des informations sur le port série et de les afficher soit en mode texte, soit en mode graphique (Fig. 4), sur l'écran de la PSP. Il nous reste maintenant à encoder, dans la couleur de chaque point, l'évolution spatiale d'une quantité physique mesurée au cours du trajet.

5 Utilisation de l'interface wifi

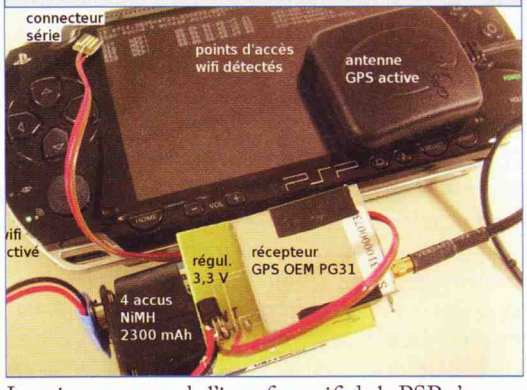
Nous avons donc démontré notre capacité à interfacer un récepteur GPS à une PSP pour afficher le trajet parcouru, et éventuellement de conserver en mémoire non volatile les latitudes et longitudes échantillonnées toutes les secondes. La PSP est par ailleurs équipée

Figure 4 : Captures d'écrans obtenues lors de l'acquisition de traces en voiture (gauche : du péage de Saint-Arnould au sud de Paris au bois de Boulogne à Paris) et à pied (droite : Jardin de Bagatelle, Paris). L'image de gauche a été recentrée en cours d'acquisition au moyen des flèches de direction à mi-hauteur alors que la trace allait sortir de l'écran : les points blancs indiquent le pas de translation lors du recentrage de la trace.



d'une interface wifi : quoi de plus naturel donc que de combiner la fonction de géolocalisation avec la capacité à écouter et identifier tous les points d'accès wifi à proximité (*wardriving* et *warwalking*) **[9]**.

Figure 5 : Montage minimal pour géopositionner les points d'accès wifi au moyen de la PSP. Dans ce cas, le connecteur entre le GPS et la PSP a été réalisé au moyen d'un bout de carte PCI (connecteurs s'enfichant normalement sur la carte mère d'un PC).



La mise en œuvre de l'interface wifi de la PSP n'est pas aisée et nécessite l'appel à des fonctions du firmware fourni par Sony et identifiées par *reverse engineering* **[10]**. La structure de données retournée par ces fonctions contient – pour un maximum de 32 points d'accès – des informations telles que les identifiants (BSSID et nom du point d'accès), la puissance reçue (RSSI, *Received Signal Strength Indication*) ou le mode d'encryption des trames. L'excellent exemple mis à disposition avec le PSPSDK dans le répertoire `samples/wlanscan_elf` nous fournit en fait pratiquement toutes les fonctionnalités dont nous avons besoin, et il nous suffit de joindre ce programme avec celui présenté précédemment pour réaliser une application de cartographie des points d'accès échantillonnés toutes les secondes (Fig. 5).

Nous avons choisi de présenter en temps réel le trajet parcouru et le nombre de points d'accès vus est codé dans la couleur du point affiché (mode graphique), ou alternativement les coordonnées de la position et les identifiants des points d'accès vus (Tab. 4).

```
 4852.6325=4887.720833 00215.2192=225.365333
015 00:03:c9:a9:9a:2e
037 00:03:c9:54:ef:e3 Wanadoo_0ffc
020 00:03:c9:55:7a:ee Wanadoo_dcb4
010 00:0f:ea:5a:7c:6f 04BTBU01
020 16:5b:a5:1e:9c:6c azur
010 00:02:2d:0b:2b:1e Airport Maison
022 00:0f:b5:ea:d4:06 MARKUS
 4852.6338=4887.723000 00215.2190=225.365000
025 00:03:c9:a9:9a:2e
010 00:14:7f:0d:36:17 SpeedTouch712E8B
027 00:03:c9:54:ef:e3 Wanadoo_0ffc
015 00:03:c9:55:7a:ee Wanadoo_dcb4
027 16:5b:a5:1e:9c:6c azur
010 00:14:bf:a5:b6:6c linksys
010 00:02:2d:0b:2b:1e Airport Maison
045 00:0f:b5:ea:d4:06 MARKUS
 4852.6341=4887.723500 00215.2198=225.366333
010 00:03:c9:54:ef:e3 Wanadoo_0ffc
032 00:03:c9:55:7a:ee Wanadoo_dcb4
017 00:0f:ea:5a:7c:6f 04BTBU01
037 16:5b:a5:1e:9c:6c azur
020 00:14:bf:a5:b6:6c linksys
012 00:02:2d:0b:2b:1e Airport Maison
 4852.6336=4887.722667 00215.2224=225.370667
005 00:03:c9:a9:9a:2e
015 00:16:cf:45:ad:b4 Livebox-9BB8
001 00:14:7f:0d:36:17 SpeedTouch712E8B
015 00:16:ce:47:dd:f2 Livebox-48B8
007 fe:64:36:5d:7f:0b freephonie
027 00:03:c9:55:7a:ee Wanadoo_dcb4
017 00:0f:ea:5a:7c:6f 04BTBU01
067 00:0f:b5:ea:d4:06 MARKUS
 4852.6336=4887.722667 00215.2260=225.376667
015 00:16:ce:47:dd:f2 Livebox-48B8
032 00:03:c9:a9:9a:2e
010 00:14:a4:28:a8:fd WANADOO-C5DA
012 00:14:7f:0d:36:17 SpeedTouch712E8B
010 fe:64:36:5d:7f:0b freephonie
032 16:5b:a5:1e:9c:6c azur
015 00:0f:ea:5a:7c:6f 04BTBU01
037 00:0f:b5:ea:d4:06 MARKUS
```

Tableau 4 : Exemple de fichier acquis à pied avec un échantillonnage toutes les secondes, présentant la position GPS – sous forme de trames ASCII brutes extraites de la transmission RS232 (format degrés, minutes et fractions de minutes) et après conversion au format degrés + fractions de degrés (validant ainsi le bon fonctionnement de la bibliothèque mathématique et du calcul flottant) – associée aux points d'accès wifi visibles identifiés dans l'ordre par la puissance reçue, le BSSID et le nom.

Figure 6 : Gauche : copie d'écran d'une trace acquise au cours d'un trajet à pied entre le bois de Boulogne et le Jardin d'Acclimatation (Paris). L'intensité de gris indique le nombre de points d'accès wifi identifiés pour chaque point GPS de la trace (échantillonnage toutes les secondes). Plus le point est clair, moins il y a de points d'accès wifi détectés. Nous constatons clairement que le nombre de points d'accès augmente lorsqu'on s'approche des quartiers résidentiels (nord) pour s'annuler dans le parc (sud). Droite : vue aérienne de la même zone. Dans cette seconde vue, l'absence de point d'accès wifi se traduit par un point vert clair qui devient d'autant plus rouge que le nombre de points d'accès détectés est élevé.

Cette présentation de l'interfaçage d'un GPS avec la PSP ne saurait être complète sans une mention au travail de développement associé à l'application Map This! disponible à http://deniska.dcemu.co.uk/. Il ne nous a malheureusement pas encore été possible de faire fonctionner ce programme avec le PG31.

Conclusion

Nous avons présenté la console de jeu Sony PlayStation Portable comme une plate-forme de développement d'applications embarquées alliant une puissance de calcul intéressante, une bonne autonomie pour un prix inférieur aux systèmes industriels équivalents. Nous avons présenté les modifications logicielles à effectuer sur la console pour la rendre apte à exécuter nos programmes (flasher un nouveau firmware), et la méthode pour installer les outils de cross-compilation sous GNU/Linux. Nous avons ensuite illustré notre capacité à écrire un programme fonctionnel en mode graphique par l'affichage de la fractale de Mandelbrot, pour ensuite aborder les aspects d'interfaçage de matériel sur le port de communication asynchrone compatible RS232. L'application finale est un outil de géolocalisation des points d'accès wifi.

Nous avons vu que le PSP SDK, basé sur gcc compilé à destination d'une architecture MIPS, nous fournit une interface de développement très proche du C classiquement utilisé pour développer des applications sous GNU/Linux. Cependant, l'utilisation intensive des fonctions fournies par le firmware propriétaire de Sony pour accéder aux fonctions bas niveau et au matériel induit certaines différences et signifie que toutes les fonctions habituellement accessibles à un programme C ne sont pas (encore) nécessairement implémentées sur PSP.

J.-M. Friedt,

friedj@free.fr

J.-M. Friedt, Association Projet Aurore, Besançon

RÉRÉRENCES

- [1] GBECG – CREMMEL (M.), « Électrocardioscope pour la Game Boy », *Elektor*, octobre 2006, p. 32-41, qui utilise la console de Nintendo en émulation Z80.
- [2] http://friedtj.free.fr/gb_eng.pdf pour quelques exemples sur Game Boy Pocket/Game Boy Classic (processeur Z80).
- [3] http://media.psp.ign.com/articles/542/542182/imgs_1.html
- [4] RAHIMZADEH (A.), « *Hacking the PSP: Cool Hacks, Mods, and Customizations for the Sony Playstation Portable* », Wiley, 2006, qui n'a que peu d'intérêt par ailleurs, puisque ne consacrant que peu de place au développement sur cette plate-forme.
- [5] PEITGEN (H.-O.), JÜRGENS (H.) & SAUPE (D.), « *Chaos and Fractals: New Frontiers of Science* », Springer, 1993.
- [6] FRIEDT (J.-M) & CARRY (É.), « Acquisition et dissémination de trames GPS à des fins de cartographie libre », *GNU/Linux Magazine France*, hors-série 27, octobre 2006.
- [7] FRIEDT (J.-M) & CARRY (É.), « Enregistrement de trames GPS – développement sur microcontrôleur 8051/8052 sous GNU/Linux », *GNU/Linux Magazine France*, 81, février 2006.
- [8] http://mc.pp.se/psp/phones.xhtml, http://nil.rpc1.org/psp/remote.html ou http://www.dcemu.co.uk/vbulletin/showthread.php?t=30035
- [9] http://en.wikipedia.org/wiki/Wardriving. Notez que je cite ici la page anglophone afin d'éviter la page francophone qui associe la recherche de points d'accès wifi à un objectif de piratage informatique que je ne cautionne pas.
- [10] http://hitmen.c02.at/files/releases/psp/syscalls.txt

Julien Boibessot

Linux Embarqué pour tous !

Voici des mots qui pourraient faire sourire pas mal de développeurs ayant déjà essayé ces petits systèmes disponibles sur le marché depuis quelque temps.
Prix trop élevés, manque d'informations ou de support pour les particuliers, attentes interminables d'extensions pour votre carte si vous n'êtes pas expert en électronique ou tout simplement marre de recycler des cartes mères PC pour réaliser vos projets...
Ces quelques points noirs rendent malheureusement l'accès au monde de Linux embarqué encore trop difficile et contraignant.
Pourtant Linux est le candidat idéal pour ce type de développement : libre, performant, stable et disposant de nombreuses extensions intéressantes, comme Xenomai (système temps réel), il a tout pour plaire !
C'est fort de ce constat que nous avons décidé de créer le « Projet Armadeus ».

1. Présentation de l'Association

Le « Projet Armadeus » est une association à but non lucratif (loi 1907, car basée en Alsace), créée dans le but de faciliter l'accès à Linux embarqué aux personnes porteuses de projets qui n'ont jusqu'alors pas abouti, faute de temps ou de moyens.

Notre objectif est aussi de favoriser la collaboration entre personnes issues d'horizons différents, afin de permettre le développement d'applications souvent irréalisables seul.

Les moyens mis en œuvre par l'association pour arriver à ces fins sont :

- mise à disposition des adhérents d'une carte de développement avec l'ensemble Logiciel libre le plus complet possible (l'achat de la carte représente les droits d'adhésion) ;
- mise en place et gestion d'une communauté de développeurs ;
- aide au développement de nouveaux projets.

Voyons en détail ces points...

1.1 Matériel

La carte APF9328 utilisée par l'association est basée sur un processeur i.MXl de la société Freescale (ex Motorola) **[1]**.

Ce processeur contient un cœur ARM9 cadencé à 192Mhz et une multitude de contrôleurs de périphériques : I2C, SPI, MMC, LCD, RS232, USB *device*, GPIO.

L'i.MXl est associé sur cette carte à une mémoire vive de type SDRAM de 16 Mo et une mémoire Flash NOR de 8 Mo, fonctionnant toutes deux sur un bus système à 96MHz. La connectivité Ethernet est assurée par un contrôleur 10/100 Mbits Davicom (DM9000).

En option, la carte peut aussi recevoir :

- un FPGA Xilinx de type Spartan 3 (200 k portes), permettant l'ajout facile de nouveaux contrôleurs de périphériques (par exemple un port PS/2, des ports PWM pour la commande de moteur, etc.). Il est même possible d'y placer un deuxième processeur (type Microblaze) !
- un CAN (Convertisseur Analogique Numérique) 8 canaux sur bus SPI et un CNA (Convertisseur Numérique Analogique) 2 canaux sur bus I2C.

L'utilisation du FPGA ne nécessite pas de connaissance particulière, car les IP (blocs fonctionnels) sont développés par plusieurs personnes du groupe spécialisées dans ce domaine.

Pour les utiliser, vous avez juste à sélectionner le ou les blocs dont vous avez besoin et les charger dynamiquement dans le FPGA. Évidemment, un driver côté Linux doit être aussi disponible pour piloter vos blocs ; son développement est en général fait conjointement avec celui de l'IP.

Pour ceux qui souhaiteraient écrire eux-mêmes le *firmware* pour ce FPGA, le kit de développement (*webpack*) est disponible gratuitement sur le site de Xilinx **[2]**.

La carte se présente sous la forme d'un petit module de 4x7cm (APF9328) qui peut venir s'enficher sur une carte de votre conception (suivant l'application) ou sur la carte de développement proposée par l'association (DevLight), permettant de tester rapidement les fonctionnalités de bases et d'en intégrer facilement de nouvelles. Cette carte dispose de connecteurs RS232, Ethernet, USB et d'alimentation et d'une zone de prototypage permettant d'accéder aux principaux signaux du module.

Figure 1 : Le module APF9328

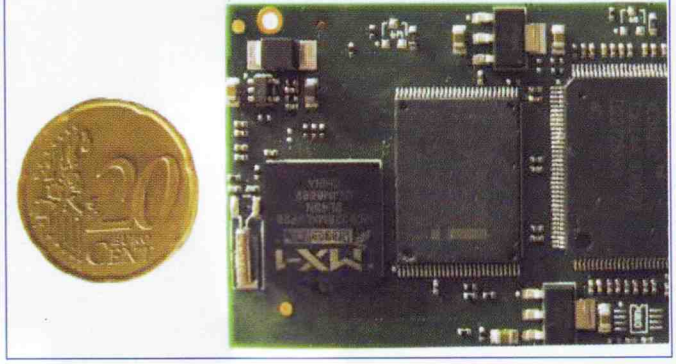


Figure 2 : La carte de développement (DevLight) avec ces connecteurs accueillant le module

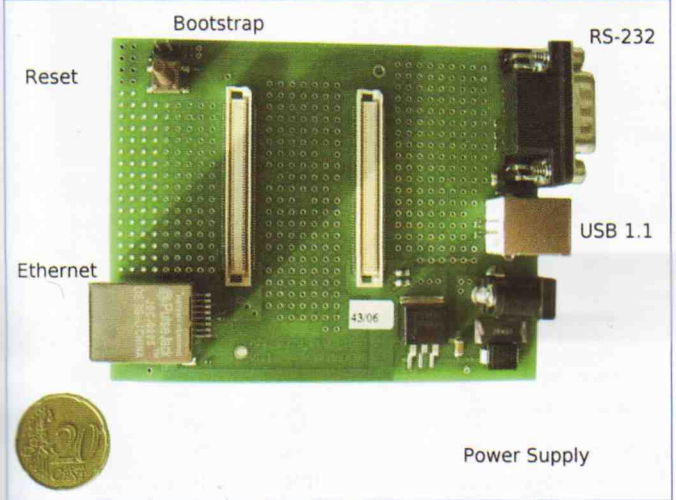
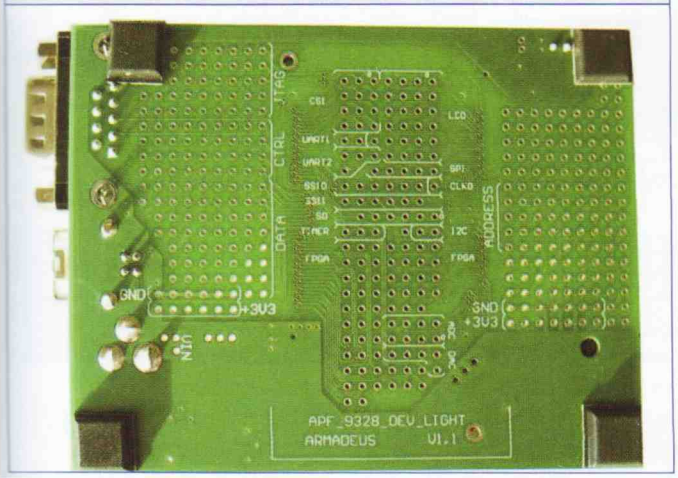


Figure 3 : La zone de prototypage sur la carte de développement avec le nom des signaux



1.2 Logiciel

1.2.1 U-Boot

Le bootloader (chargeur de démarrage) du système est U-Boot **[3]**. C'est lui qui :

- initialise les fonctions vitales du système (horloge, SDRAM) après un reset ;
- fournit une interface permettant de reprogrammer la Flash du système (après chargement série ou Ethernet/TFTP) ;
- charge le code du FPGA au démarrage (si besoin) ;
- lance le système Linux (ou vos programmes maison ne nécessitant pas d'OS), à partir de la Flash ou bien d'une connexion réseau (TFTP).

1.2.2 Linux

L'OS du système est une version 2.6.18 de Linux, adaptée au processeur i.MXl. Cette version contient les pilotes pour les périphériques suivants : SPI, I2C, Ethernet, chargement FPGA, PWM, carte mémoire SD/MMC, clavier PS/2, écrans LCD graphiques sans contrôleurs (320x240 TFT & STN), port parallèle compatible ppdev (permettant de contrôler des LCD textes compatibles LCD4Linux) et USB device (Mode *Mass Storage* et Série).

1.2.3 uClibc/Buildroot

Le système de fichier (rootfs) est généré par Buildroot **[4]** et toutes les commandes Unix standards sont supportées grâce au « couteau suisse » de l'embarqué : Busybox **[5]**.

Vous pouvez créer des programmes C/C++ Unix standard grâce à la présence de la bibliothèque C pour l'embarqué : uClibc **[6]**. L'i.MXl disposant d'une MMU (*Memory Management Unit*), vous avez la même interface de programmation que sur un système Linux x86 classique ; les espaces mémoire noyau et processus utilisateurs sont distincts, a contrario des systèmes uCLinux à base de processeurs ARM7.

1.2.4 Applicatif

L'utilisateur a la possibilité d'utiliser Qtopia Core **[7]** ou SDL **[8]** pour développer ses applications (graphiques ou non) de manière portable.

L'avantage de telles API réside dans le fait que l'application peut être entièrement écrite et simulée sur le PC de développement (Linux ou autre).

Si tant est qu'une bibliothèque ne soit pas supportée par Buildroot, il est assez facile d'intégrer celle-ci dans le processus de construction du système à partir du moment où cette bibliothèque supporte le classique `configure`, `make`, `make install` et la compilation croisée.

1.3 Support utilisateur

Le support utilisateur du projet s'articule autour de 4 points :

- un *wiki* **[9]** basé sur le moteur MediaWiki (aussi utilisé par Wikipédia) qui fournit un support complet aux utilisateurs sur toutes les fonctionnalités du système. Il leur permet aussi de participer à l'amélioration du contenu, soit en modifiant les pages existantes, soit en ajoutant leurs propres pages.
- un gestionnaire de bug : eh oui, tout projet contenant du logiciel se doit d'en posséder un ;-)
- une *mailing list* qui permet de faciliter la communication entre les différents développeurs. Les points techniques et organisationnels sont discutés ici.
- l'accès aux sources : nous utilisons SVN comme gestionnaire de version. L'accès en écriture à la base est réservé aux « intégrateurs », mais n'importe qui peut charger les sources sur SourceForge **[10]** et nous faire part de ses modifications sous forme de patches (principe similaire à U-Boot et Linux).

1.4 Limitations

Étant une association à but non lucratif, nous avons limité l'utilisation des cartes obtenues dans le cadre de l'association à un usage particulier ou éducatif (autre association ou établissement d'enseignement). De même, les applications réalisées au sein du projet doivent être placées sous licence libre de manière à ce que tous les membres puissent en profiter et de manière à garantir

leur pérennité. Nous n'avons pas non plus pour but la promotion de telle ou telle entreprise, mais bel et bien celle des Logiciels libres dans l'embarqué, vis à vis du plus grand nombre.

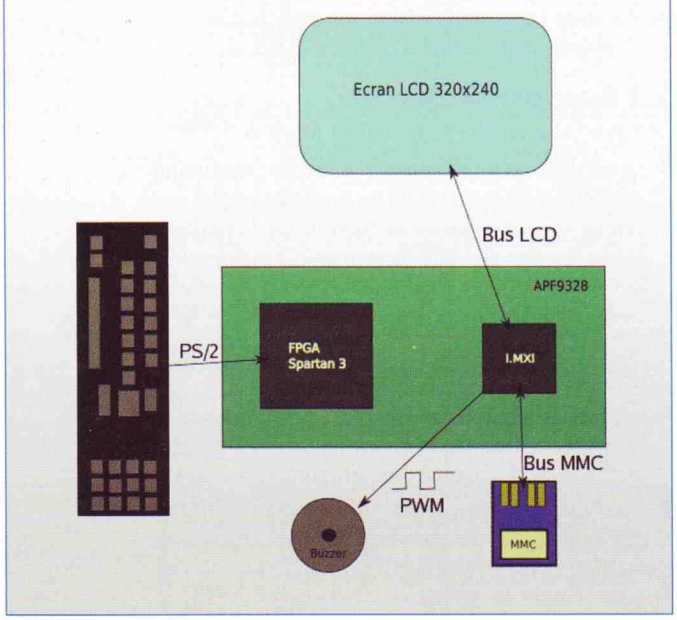
2. Exemple d'application

Voilà, après cette présentation, qui, je l'espère, ne vous a pas semblé trop rébarbative, voyons concrètement quel type d'application nous pouvons construire à partir des briques de base fournies par le projet. Vous vous êtes toujours dit que développer une console de jeu portable, c'était compliqué. Eh bien, détrompez vous, nous allons réaliser un prototype qui n'aura pas grand-chose à envier à ses concurrentes commerciales (hormis la logithèque ;-)).

Pour construire cette console, nous aurons besoin :

- d'un *keypad* : nous utiliserons un clavier PS/2 (pour notre première maquette) ;
- d'un écran LCD : nous utiliserons un écran TFT de récupération (Sharp LQ057 avec une résolution de 320x240 et des couleurs sur 18bits) ;
- d'un lecteur de carte mémoire pour stocker les jeux ; nous utiliserons une MMC comme « cartouche de jeu » ;
- d'une sortie son, rudimentaire dans notre exemple, obtenue avec la sortie PWM de l'i.MXl.

Figure 4 : Diagramme fonctionnel du système



2.1 Installation des outils de développement

Nous avons, dans un premier temps, besoin d'installer tous les outils nécessaires au développement. La génération du compilateur croisé, du débogueur, des images noyau et du système de fichier est entièrement automatisée, grâce à Buildroot. L'utilisateur doit juste disposer d'une connexion réseau haut-débit, car le script d'installation va aller charger tous les fichiers requis sur Internet. L'installation fonctionne sur Linux et Window$/Cygwin, mais, dans notre cas, nous utiliserons notre système préféré.

La première chose à faire est de récupérer l'archive de la dernière version du logiciel Armadeus sur SourceForge : http://sourceforge.net/projects/armadeus/.

Supposons que vous ayez chargé l'archive dans votre répertoire `$HOME`. Il faut ensuite la décompresser :

```
$ cd $HOME
$ tar jxf armadeus-2.0.tar.bz2
```

NOTE

Assurez-vous d'avoir au moins 2 Go d'espace libre à l'endroit où vous décompressez l'archive, car la compilation des outils de développement est assez gourmande en ressources !!

Un répertoire `armadeus-2.0` est créé à l'endroit de la décompression. Il contient tous les fichiers nécessaires à la construction de votre environnement de développement.

Ensuite, il faut vérifier que les utilitaires nécessaires au processus sont bien présents sur votre système (par exemple sous Debian) :

```
$ sudo apt-get install autoconf automake bison ckermit \
  flex g++ gettext libncurses5-dev subversion texinfo \
  tftpd wget zlib1g-dev
```

Une fois ces vérifications effectuées, lancez la compilation :

```
$ cd armadeus-2.0
[armadeus-2.0]$ make menuconfig
```

Un système de configuration pseudo-graphique semblable au `menuconfig` du noyau Linux apparaît. Laissez les paramètres par défaut pour l'instant et enregistrez la configuration ([Escape], puis Yes). Enfin, lancez :

```
[armadeus-2.0]$ make
```

Quelques heures plus tard (suivant votre système ;-)), votre environnement est prêt. Dans le répertoire `buildroot/`, vous devriez voir les fichiers suivants :

- `u-boot.bin`, l'image U-Boot ;
- `linux-kernel-2.6.18.1-arm.bin`, le noyau Linux ;
- `rootfs.arm_nofpu.jffs2`, l'image du système de fichier.

Il nous faut maintenant ajouter, dans la variable d'environnement `PATH`, le chemin vers les outils de compilation, afin par exemple de pouvoir lancer le compilateur croisé en appelant juste `arm-linux-gcc`. Rajoutez donc dans votre `~/.bashrc` la ligne suivante :

```
export PATH=$PATH:/emplacement_archive/armadeus-2.0/
buildroot/build_arm_nofpu/staging_dir/bin
```

2.2 Programmation de la carte

Une fois les outils de compilation et les images système générées, il vous faut les « flasher » sur la carte. Pour cela, nous allons utiliser U-Boot et la connexion réseau entre la carte et votre PC.

LISEZ-VOUS RÉGULIÈREMENT :

OFFRES DE COUPLAGE

Le magazine 100 % Linux

et / ou

Le magazine 100 % Sécurité

et / ou

100 % PRATIQUE

et / ou

Apprivoisez votre pingouin !

SI OUI, ALORS CES OFFRES D'ABONNEMENT À TARIF PRÉFÉRENTIEL VOUS SONT DESTINÉES...

(1) Pour 11 N°s Linux Magazine + 6 N°s Linux Mag HS - (2) Pour 11 N°s Linux Magazine + 6 N°s MISC - (3) Pour 11 N°s Linux Magazine + 6 N°s MISC + 6 N°s Linux Mag. HS - (4) Pour 11 N°s Linux Magazine + 6 N°s MISC + 6 N°s Linux Mag. HS + 6 N°s Linux Pratique

OFFRE DE COUPLAGE À REMPLIR ET À RETOURNER À (OU PHOTOCOPIE)

LINUX MAGAZINE - BP 20142 - 67603 SELESTAT CEDEX

☐ **OUI, je m'abonne et désire profiter des offres spéciales de couplage**

Référence de l'offre :	Prix	Qté.	Total
☐ 11 N°s Linux Mag. + 6 N°s Linux Mag HS	79 €		
☐ 11 N°s Linux Mag. + 6 N°s MISC	83 €		
☐ 11 N°s Linux Mag. + 6 N°s MISC + 6 N°s Linux Mag HS	105 €		
☐ 11 N°s Linux Mag. + 6 N°s MISC + 6 N°s Linux Mag HS + 6 N°s Linux Pratique	129 €		
OFFRES VALABLES UNIQUEMENTS EN FRANCE MÉTRO.		TOTAL	

Pour les tarifs étrangers, consultez notre site : www ed-diamond.com

LES 4 FAÇONS DE VOUS ABONNER !

» PAR COURRIER POSTAL EN NOUS RENVOYANT LE BON CI-DESSOUS.

» PAR LE WEB, SUR NOTRE SITE : WWW.ED-DIAMOND.COM.

» PAR TÉLÉPHONE (PAIEMENT C.B.) ENTRE 9H-12H & 15H-16H AU 03 86 56 02 06.

» PAR FAX AU 03 86 56 02 09 C.B. ET/OU BON DE COMMANDE ADMINISTRATIF

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon réglement :

☐ Je règle par chèque bancaire ou postal à l'ordre de Diamond Editions

☐ Paiement par carte bancaire :

N° Carte :

Expire le : Cryptogramme Visuel : Voir image ci-dessous

Date et signature obligatoire : 200

Boostez votre collection !

Avez-vous L'ÂME du COLLECTIONNEUR ?

VOUS RECHERCHEZ UN MAGAZINE EN PARTICULIER? ALLEZ SUR WWW.ED-DIAMOND.COM POUR VOIR LE SOMMAIRE DÉTAILLÉ DE CHAQUE MAGAZINE ET ENSUITE...
BOOSTEZ VOTRE COLLECTION AVEC LES "POWER PACKS X5", SOIT 5 LINUX MAGAZINE POUR 15€ ET LES "POWER PACKS X10", SOIT 10 LINUX MAGAZINE POUR 25€ À CHOISIR DANS LA LISTE CI-DESSOUS :

LES 4 FAÇONS DE COMMANDER !

- PAR COURRIER POSTAL EN NOUS RENVOYANT LE BON CI-DESSOUS.
- PAR LE WEB, SUR NOTRE SITE : WWW.ED-DIAMOND.COM.
- PAR TÉLÉPHONE (PAIEMENT C.B.) ENTRE 9H-12H & 15H-18H AU 03 88 58 02 08.
- PAR FAX AU 03 88 58 02 09 C.B. ET/OU BON DE COMMANDE ADMINISTRATIF

Choisissez vos numéros dans le tableau ci-dessous*

***SEULS LES NUMÉROS CI-DESSOUS SONT DISPONIBLES POUR UNE COMMANDE DE POWER PACKS PAR X5 ET X10**

- N°06 GNOME - The Gimp
- N°07 Dopez Linux
- N°08 Le futur résolument objet
- N°09 Prêt pour le jeu !
- N°10 The HURD : 100% GNU
- N°11 Exclusif : l'avenir de G.N.O.M.E
- N°12 NT et Linux : Guerre ou complément ?
- N°13 Cryptage : la clé de la sécurité
- N°14 XFree 4.0 : le futur à notre portée
- N°15 Passez à la vitesse supérieure
- N°16 OpenSources : Est-ce suffisant?
- N°17 Linux : Système embarqué
- N°18 Spécial interview : l'avenir de Linux
- N°19 Dossier spécial : Postgre SQL 7.0
- N°21 Le protocole Internet du 21e siècle : IPv6
- N°22 Le multi-hreading : Une manière moderne de programmer le Multitâche
- N°23 Débugger sous Linux
- N°24 Palm et Linux
- N°25 Kernel 2.4.0
- N°26 <Dossier> XML </Dossier>
- N°27 Les systèmes de fichiers journalisés
- N°28 Scripting : la force d'Unix
- N°29 L.F.S. Linux From Scratch
- N°30 Le chiffrement des données
- N°31 VPN et tunneling
- N°32 Changez de coquille
- N°34 XSL - FO : TeX Killer ?
- N°35 QoS et iproute : optimisation et contrôle du trafic IP
- N°36 Linux embarqué : Le projet mGlinux
- N°37 L'impression sous Linux
- N°38 Le desktop Shell : Enlightenment
- N°39 Sécurité : Patchez votre noyau !
- N°40 MySQL : la base de donnée OpenSource
- N°41 Steganographieou l'art de la dissimulation de données
- N°42 Développez vos pilotes de périphérique
- N°43 Administrez facolement votre réseau SNMP
- N°44 Comprenez NetBios pour Maîtriser l'interopérabilité windows GNU Linux
- N°45 Cohabitation : UnDNS Bind dans un reseau Windows 2000
- N°46 Debian : Utilisez Samba avec le support ACL
- N°47 GNUstep : le petit frère de Mac OS X ?
- N°48 Caudium, votre prochain serveur Web !
- N°49 Après MySQL & PostgreSQL SAP DB : La base de données libre & puisante
- N°50 Créez un album Photo avec PHP ...et sans MySQL
- N°51 Boostez votre site Web avec XML grâce à XSLT, CSS & XPath
- N°52 Linux Temps réel où en est-on aujourd'hui ?
- N°53 Linux sur PDA : Linux dans votre poche !
- N°54 Maîtrisez LVM
- N°55 Intelligence Artificielle : Principes & programmation de jeux de stratégie classique
- N°56 Développez vos applications Mozilla avec XPFE & XPCOM
- N°57 Maîtrisez la gestion... Slots & Signaux ... des évènements en C++
- N°58 Djbdns enfin une alternative viable à BIND !
- N°59 Zopix, Créez un CD "Live" Zope en 10 minutes !
- N°60 JBoss serveur d'applications J2EE OpenSource
- N°61 Découvrez MySQL 5 et les procédures stockées
- N°62 Créez votre OS, principe et implémentation
- N°63 Les threads : kernel 2.6 et 2.4
- N°64 Adamoto
- N°65 Théorie et pratique : Supervision avec Nagios
- N°66 Créez votre Distribution Live
- N°67 C# .NET
- N°68 Le crash disque vous guette
- N°69 La réponse de Sun à Linux ? SOLARIS 10
- N°70 Découvrez et comprenez la technologie GRID
- N°71 Présentation et installation du Hurd
- N°72 Services Web... C/C++ et gSOAP
- N°73 Compression théorie algorithmes et programmation
- N°74 VFS : Système de fichiers virtuel
- N°75 Tuning de code
- N°76 Algorithmes évolutionnistes
- N°77 Systèmes de fichiers chiffrés
- N°78 Bluetooth
- N°79 Sécurisation du Noyau avec PAX
- N°80 Run in memory
- N°81 Comment fonctionnent les générateurs de nombres pseudo-aléatoires

NUMÉROS LINUX MAGAZINE ÉPUISÉS
N°01, N°02, N°03, N°04, N°05, N°20, N°33.

BON DE COMMANDE POWER PACKS À REMPLIR ET À RETOURNER À (OU PHOTOCOPIE)

LINUX MAGAZINE - BP 20142 - 67603 SELESTAT CEDEX

	OUI, je désire acquérir un POWER PACK X5	1er 1PP* X5	2ème 2PP* X5	3ème 3PP* X5
Cochez ici ▶ POWER PACKS X5	1, Linux Magazine N°			
	2, Linux Magazine N°			
	3, Linux Magazine N°			
	4, Linux Magazine N°			
	5, Linux Magazine N°			
	Total par série de POWER PACKS X5 :	15 €	30 €	45 €
	OUI, je désire acquérir un POWER PACK X10	1er 1PP* X10	2ème 2PP* X10	3ème 3PP* X10
Cochez ici ▶ POWER PACKS X10	1, Linux Magazine N°			
	2, Linux Magazine N°			
	3, Linux Magazine N°			
	4, Linux Magazine N°			
	5, Linux Magazine N°			
	6, Linux Magazine N°			
	7, Linux Magazine N°			
	8, Linux Magazine N°			
	9, Linux Magazine N°			
	10, Linux Magazine N°			
	Total par série de POWER PACKS X10 :	25 €	50 €	75 €
	Les Hors Séries et numéros spéciaux sont exclus des POWER PACKS. Montant TOTAL 15€ + 3,81€ de frais de port. Le TOTAL s'élève à 18,81€ pour l'achat d'un POWER Pack x5.	TOTAL :		
	SEULEMENT EN FRANCE MÉTROPOLITAINE !	Frais de port :		+3,81€
	*PP= POWER PACK	TOTAL :		

1 Voici mes coordonnées postales

Nom :

Prénom :

Adresse :

Code Postal :

Ville :

2 Je joins mon réglement :

☐ Je règle par chèque bancaire ou postal à l'ordre de Diamond Editions

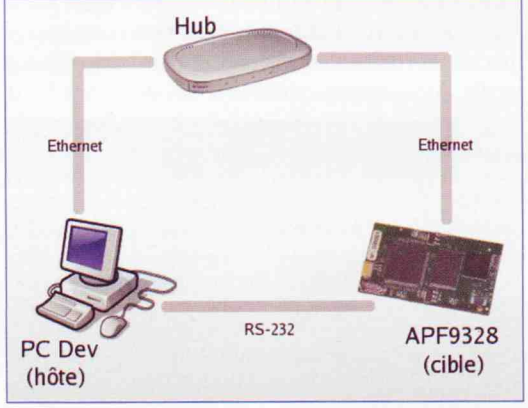
☐ Paiement par carte bancaire :

N° Carte :

Expire le : Cryptogramme Visuel : Voir image ci-dessous

Date et signature obligatoire : 200

Figure 5 : Schéma montrant comment la cible et l'hôte doivent être interconnectés lors du développement



U-Boot utilise le protocole TFTP **[11]** pour transférer les données sur Ethernet, donc nous allons nous assurer que votre hôte est capable de le gérer correctement. Créez un répertoire `/tftpboot` en tant que *root* et donnez lui les droits en lecture écriture pour tout le monde :

```
$ sudo /bin/bash
# mkdir /tftpboot
# chmod 777 /tftpboot
# exit
```

Configurez ensuite le démon tftpd pour qu'il utilise ce répertoire (référez-vous à la doc de votre distribution ou au wiki de l'association (généralement cela passe par l'édition du fichier `/etc/xinetd.d/tftp`) et copiez les images générées par la compilation dans `/tftpboot/`.

```
[armadeus-2.0]$ cp buildroot/u-boot.bin /tftpboot
[armadeus-2.0]$ cp buildroot/linux-kernel-2.6.18.1-arm.bin /tftpboot
[armadeus-2.0]$ cp buildroot/rootfs.arm_nofpu.jffs2 /tftpboot
```

Connectez ensuite votre carte sur un port série de votre PC (avec un câble croisé/Null Modem) et sur votre réseau Ethernet local. Le port série vous permet de communiquer avec les consoles U-Boot et Linux. Pour cela, vous devez avoir auparavant installé et configuré un émulateur de terminal série (nous recommandons C-Kermit **[12]** ; référez-vous au wiki si jamais vous avez besoin de plus d'infos pour l'installer).

Lancez Kermit afin de voir ce que la carte va vous envoyer sur la console série :

```
$ kermit -c
Connecting to /dev/ttyUSB0, speed 115200
 Escape character: Ctrl-\ (ASCII 28, FS): enabled
Type the escape character followed by C to get back,
or followed by ? to see other options.
----------------------------------------------------
```

NOTE

Dans mon cas, kermit est configuré pour utiliser mon adaptateur USB<->Série, associé au device `/dev/ttyUSB0`.

2.3 Démarrage de la carte

Mettez la carte sous tension : comme tout système embarqué, c'est le *bootloader* (ici U-Boot) qui démarre en premier. Celui-ci configure le processeur afin qu'il soit capable d'utiliser sa mémoire et d'accéder au périphérique de boot. U-boot a aussi la particularité d'offrir une console série à l'utilisateur afin que celui-ci puisse effectuer certaines actions manuellement.

Une fois le système initialisé, le bootloader charge le noyau Linux en mémoire. Il peut aller chercher l'image du noyau soit sur la mémoire Flash de la carte, soit sur le réseau. Le bootloader passe ensuite la main au noyau qui s'auto-décompresse.

Linux démarre, initialise ses périphériques, essaie de monter son système de fichier et ensuite exécute le programme `/sbin/init`. Ensuite, l'initialisation est dépendante du contenu du fichier `/etc/inittab`. À la fin de l'initialisation, vous avez en général une console Linux sur le port série, certains services réseau démarrés et si vous avez branché un LCD, des terminaux virtuels sur l'écran. Un peu comme sur les systèmes Linux des grands ;-).

Pour reflasher vos images, interrompez le démarrage de U-Boot en appuyant sur une touche de votre clavier lorsque celui-ci vous le propose.

```
U-Boot 1.1.3 (Oct 26 2006 - 20:27:59) ap9328 patch 2.4
U-Boot code: 0BF80000 -> 0BFA2524  BSS: -> 0BFA72D4
RAM Configuration:
Bank #0: 08000000 16 MB
Flash:  8 MB
In:    serial
Out:   serial
Err:   serial
Hit any key to stop autoboot:  0
BIOS>
```

A supposer que votre serveur tftp soit bien installé et vos images présentes sur celui-ci, vous pouvez lancer les commandes suivantes pour mettre à jour le noyau Linux et le rootfs :

```
BIOS > tftp 8000000 linux-kernel-2.6.18.1-arm.bin
BIOS > run flash_kernel
BIOS > tftp 8000000 rootfs.arm_nofpu.jffs2
BIOS > run flash_rootfs
```

NOTE

`tftp` est la commande U-Boot qui permet de télécharger sur la cible, par le protocole TFTP, des données depuis votre hôte. Ces données sont placées dans la RAM de la cible, ici à l'adresse 0x08000000. `flash_kernel` et `flash_rootfs` sont deux scripts U-Boot permettant de flasher une image chargée en mémoire ; ils sont exécutés grâce à la commande `run` de U-Boot.

Votre système étant à jour, redémarrez la carte (bouton *reset*) et laissez Linux démarrer. Vous devriez voir apparaître le *login* système sur la console série :

```
Welcome to the Armadeus project.
armadeus login:
```

2.4 Configuration matérielle

Maintenant que vous savez générer un système fonctionnel pour votre carte, nous allons ajouter le logiciel pour la transformer en console de jeu.

Le processeur i.MXl utilisé sur la carte Armadeus étant très souple au niveau de la configuration des périphériques, vous avez la possibilité de choisir d'activer certains modules hardware ou pas et ainsi d'avoir une configuration personnalisée. En désactivant des périphériques, vous récupérez ainsi des entrées/sorties généralisées (GPIO) que vous pouvez utiliser pour votre application. Par exemple, imaginons que vous n'ayez pas besoin du contrôleur LCD et que d'un seul port série. Vous pouvez les désactiver et ainsi avoir à votre disposition un vingtaine de pattes du processeur en tant qu'entrée/sortie.

Nous sommes en train de réfléchir à un outil graphique permettant de configurer le processeur de manière fine en quelques clics, mais pour l'instant la configuration matérielle se fait plus classiquement en activant ou non les drivers lors de la compilation du noyau Linux :

```
[armadeus]$ make linux-menuconfig
```

Dans la section « Device Drivers », activez :

- le LCD Sharp LQ057 dans *Graphic support->Motorola i.MX LCD support* ;
- la PWM (pour le son) et le PS/2 dans *Armadeus Specific drivers ->* ;

(le driver MMC est activé par défaut dans la configuration standard)

Sauvez votre configuration.

Figure 6 : Sélectionnez les drivers nécessaires

```
    LED devices  --->
    Multimedia devices  --->
    Graphics support  --->
    Sound  --->
    USB support  --->
    Armadeus specific drivers  --->
    MMC/SD Card support  --->
    Real Time Clock  --->
```

La connexion physique entre ces périphériques et la carte est détaillée sur le wiki du projet **[13]**. Je vais essayer de la résumer simplement. Le LCD est branché directement sur les signaux venant du contrôleur intégré à l'i.MXl. Généralement un écran TFT sans contrôleur se pilote avec des signaux de contrôle/horloge (`clock`, `hsync`, `vsync`, `enable`) et des signaux de données représentant les valeurs de chaque composante de couleur pour un pixel donné. Dans le cas du LCD Sharp, chaque composante (Rouge, Verte, Bleue) peut être codée sur 6 bits, d'où la dénomination LCD 18 bits (3x6). L'i.MX ne pouvant piloter que des LCD au maximum 16 bits, il faut relier deux des lignes de poids faible de deux composantes à la masse (dans notre exemple Rouge et Bleu). On perd ainsi quelques couleurs par rapport aux possibilités de l'écran, mais il nous en reste encore plus de 65000 !

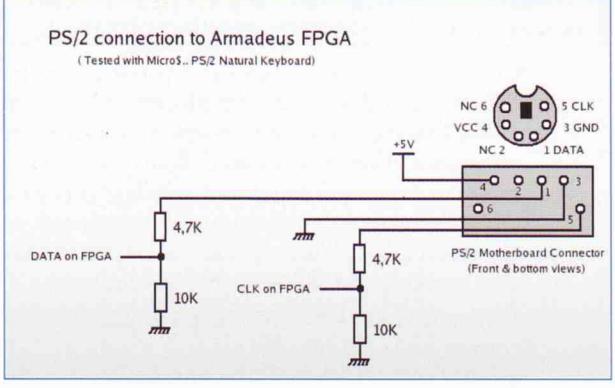
La logique du LCD est alimentée directement par le 3,3 V de la carte. Par contre, pour le rétro-éclairage une tension de 12 V est nécessaire : j'utilise ma bonne vieille alimentation stabilisée pour cela.

NOTE

Notez qu'au moment où vous lirez ces lignes, l'association proposera à ses membres un LCD TFT Microtips 320x240 bon marché, avec sa carte d'adaptation ne nécessitant pas de tension externe pour le rétro-éclairage.

L'i.MXl ne disposant pas de contrôleur PS/2 en standard, nous implanterons celui-ci dans le FPGA. Ce dernier est relié à l'i.MXl par l'intermédiaire d'un bus 16 bits. Ainsi, on peut accéder aux IP programmées dans le FPGA comme s'il s'agissait d'un coprocesseur. Il est aussi possible de programmer le FPGA directement depuis l'i.MX, sans avoir recours à une sonde JTAG ou une mémoire externe. Les données sont stockées sur la Flash du système, lues par l'i.MX et envoyées au FPGA en série sur une GPIO, par U-Boot ou bien sous Linux. Pour notre IP PS/2, le FPGA nous fournit les 2 signaux CLOCK et DATA qui peuvent être directement reliés au connecteur du clavier si celui-ci accepte d'être alimenté en 3,3 V. Dans mon cas (clavier PS/2 + USB), il m'a fallu alimenter le clavier en 5 V et donc ajouter un régulateur de tension supplémentaire et un pont diviseur entre ce dernier et les signaux du FPGA.

Figure 7 : Adaptation des signaux PS/2 (5 v) vers ceux du FPGA (3,3 v)



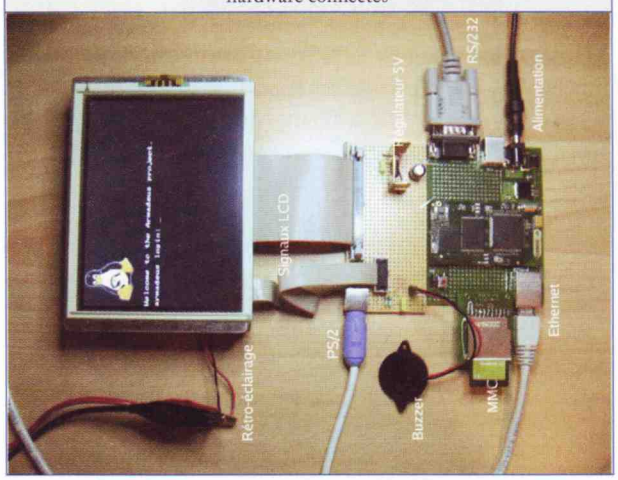
La carte ne disposant pas d'un contrôleur sonore, nous utiliserons dans un premier temps la sortie PWM **[14]** de l'i.MXl qui est largement suffisante pour générer des petits bruitages (une carte d'extension avec *chip* sonore est actuellement en cours de développement). La PWM sera reliée directement à un petit *buzzer* piézoélectrique.

NOTE

Il est possible d'améliorer la qualité du son en plaçant un filtre passe-bas en sortie de la PWM, transformant ainsi le signal carré en sinusoïde, qui est de meilleur rendu sur un buzzer.

Pour la MMC, l'i.MXl disposant d'un contrôleur intégré, il suffit de connecter les signaux nom pour nom. Pour le connecteur, vous avez le choix entre bricoler le vôtre ou bien utiliser celui proposé aux membres de l'association. Celui-ci vient avec un petit PCB sur lequel on vient simplement connecter les signaux de la carte de développement.

Figure 8 : Voici à quoi ressemble le prototype une fois tous les éléments hardware connectés



2.5 Configuration logicielle spécifique

Maintenant que le hardware est connecté et que les drivers Linux sont activés, ajoutons les couches de « haut niveau ». Dans notre exemple, seule SDL sera nécessaire en plus de la configuration par défaut, donc lancez le configurateur de Buildroot et ajoutez cette bibliothèque :

```
[armadeus-2.0]$ make menuconfig
```

Figure 9 : Ajouter des packages dans buildroot

```
uild options  --->
oolchain Options  --->
Package Selection for the target  --->
arget Options  --->
```

Relancez la génération du système :

```
[armadeus-2.0]$ make
```

Quelques minutes plus tard, nos nouvelles images système sont prêtes. Nous pouvons les reflasher comme expliqué précédemment (2.2).

Reste encore à faire en sorte que le FPGA soit chargé avec l'IP PS/2 à chaque démarrage, par U-Boot. Pour cela, copier le fichier de configuration du FPGA dans `/tftpboot` :

```
[armadeus-2.0]$ cp firmware/PS2/ps2_top.bit /tftpboot/
```

Passez en mode console sous U-Boot et chargez le fichier en mémoire :

```
BIOS> tftp 8000000 ps2_top.bit
```

Puis flashez-le et activez le chargement automatique au démarrage :

```
BIOS> run flash_firmware
 .. done
 Erased 2 sectors
 Copy to Flash... done
 Flashing Firmware succeed
BIOS> setenv firmware_autoload 1
BIOS> saveenv
```

2.6 Le jeu

Pour notre premier jeu, nous ferons simple : nous allons faire un clone d'Arkanoid/Breakout en C. Les sources du jeu sont incluses dans le projet, dans le répertoire `target/demos/armanoid/`.

Pour compiler la version PC, faites :

```
[armadeus-2.0]$ make -C target/demos/armanoid/
```

Et pour la version pour la cible :

```
[armadeus-2.0]$ make -C target/demos/armanoid/ clean
[armadeus-2.0]$ make -C target/demos/armanoid/ TARGET=arm
```

Le but de cet article n'étant pas d'apprendre à utiliser SDL, nous partons du principe que vous avez déjà suivi les précédents articles à ce sujet dans GLMF.

Le code du jeu en lui-même n'étant pas une référence en matière de programmation de jeu, pour ceux qui connaissent déjà SDL, voire en matière de programmation tout court ;-), nous nous attarderons juste sur le module `buzzer.c` qui permet de jouer des sons depuis notre jeu afin de le rendre un peu plus sympa.

2.6.1 Production du son

Ce module est en fait une mini-bibliothèque avec 3 fonctions principales :

- `initBuzzer()`, qui permet d'initialiser le système ;
- `playSound( freq, temps )`, qui permet de jouer un son d'une fréquence donnée pendant un certain temps ;
- `releaseBuzzer()`, qui permet de libérer toutes les ressources utilisées.

```
...
#define FREQ_SYS_FILE   "/sys/class/pwm/pwm0/frequency"
#define ACTIVE_SYS_FILE "/sys/class/pwm/pwm0/active"
...
int initBuzzer()
{
   int result = -1;
   if( !initialized )
   {
       playing = 0;
       fd  = fopen( FREQ_SYS_FILE, "w" );
       fda = fopen( ACTIVE_SYS_FILE, "w" );

       if( (fd != NULL) && (fda != NULL) )
       {
           result = 0;
           initialized = 1;
       } else
           printf("Buzzer: pb pour ouvrir /sys/...\n");
   }
   return( result );
}
```

Pour jouer un son, nous utilisons donc le driver PWM. Celui-ci se commande très simplement depuis l'interface fichier `/sys` :

- `/sys/class/pwm/pwm0/active`, lorsque l'on écrit 1 dans ce fichier la PWM se met en marche. (0 pour l'arrêter) ;
- `/sys/class/pwm/pwm0/frequency`, une fois démarrée, on peut mettre dans ce fichier la fréquence désirée.

Par exemple, pour jouer un son depuis un *shell*, nous ferions :

```
# echo 1 > /sys/class/pwm/pwm0/active
# echo 500 >  /sys/class/pwm/pwm0/frequency
... on attend un petit peu ...
# echo 0 > /sys/class/pwm/pwm0/active
```

Dans le cas de notre bibliothèque, cela revient à ouvrir les fichiers avec `fopen()`, puis à faire des `fwrite()` dessus.

```
void releaseBuzzer()
{
   if( initialized )
   {
       playing = 0;
       SDL_RemoveTimer( myTimerID );
       timerExpired(0, 0);
       fclose( fd );
       fclose( fda );
       initialized = 0;
   }
}

void playSound( int aFreq, int aTime )
{
   if( (!playing) && (initialized) )
   {
       char freq[16]; int nchar;

       // Limite la frequence entre [50,15k] Herz
       if( aFreq < 50 ) aFreq = 50;
       if( aFreq > 15000 ) aFreq = 15000;
       // Démarre la PWM
       fwrite( "1", 1, 1, fda );
       fflush( fda );
       // Configure la PWM avec la fréquence requise
       nchar = sprintf( freq, "%d", aFreq);
       fwrite( freq, nchar, nchar, fd );
       fflush( fd );
       // Lance un timer qui stoppera le son
       // une fois le délai expiré
       myTimerID = SDL_AddTimer( (Uint32) aTime, \
                                 timerExpired , 0);
       playing = 1;
   }
}
```

La fonction `playSound()` utilise les *timers* SDL afin de planifier la durée pendant laquelle le son doit être joué. À noter qu'après chaque `fwrite()` dans l'interface fichier de la PWM, il faut faire un `fflush()`, afin de signaler au système d'effectuer l'écriture immédiatement, sinon celle-ci est mise dans un tampon.

2.6.2 Préparation de la SD/MMC

La plupart des SD/MMC commercialisées ne sont pas formatées correctement ou ne disposent pas de table de partition standard. Pour pouvoir utiliser correctement une MMC sur la carte Armadeus, nous vous recommandons :

- d'effacer toute partition pouvant être présente sur la carte et de créer un partition primaire unique (type Linux ou Window$), en utilisant fdisk et un lecteur de carte sur votre PC de développement ;
- de formater ensuite la partition en EXT2 ou VFAT, selon vos besoins.

Nous ne détaillerons pas les étapes (rébarbatives) de cette « mise à neuf » de la MMC. Pour ceux qui auraient besoin de plus de renseignements, tout est expliqué sur le wiki de l'association **[15]**.

À supposer que la carte est désormais prête à être utilisée (dans mon cas, associée à `/dev/sdd1`) et qu'il existe un point de montage `/mnt/mmc` sur votre hôte, copions maintenant Armanoid du PC sur la MMC :

```
# mount /dev/sdd1 /mnt/mmc/
$ cp target/demos/armanoid/armanoid /mnt/mmc/
$ cp target/demos/armanoid/*.bmp /mnt/mmc/
# umount /mnt/mmc
```

2.6.3 Lancement du jeu sur la cible

Nous allons maintenant écrire un petit script qui va mettre en place tout le système à chaque démarrage.

Pour cela, nous allons rajouter un élément dans le répertoire `/etc/init.d/` de la carte. Ce répertoire contient les scripts lancés automatiquement à chaque démarrage de la carte (scripts commençant par « S »). Ajoutons donc, dans le `rootfs`, le nôtre qui sera lancé en dernier :

```
# vi /etc/init.d/S99game

  #!/bin/sh

  # Chargement des modules nécessaires:
  modprobe pwm
  modprobe apf9328ps2
```

N'oublions pas de rendre le script exécutable :

```
# chmod a+x /etc/init.d/S99game
```

Redémarrez la carte Armadeus :

```
# reboot
```

et attendez l'invite de login sur l'écran. Insérez votre MMC, celle-ci est reconnue automatiquement :

```
imx-mmc imx-mmc.0: card inserted
mmcblk0: mmc0:0001 MMC    250880KiB
 mmcblk0: p1
```

Montons la MMC dans `/media/mmc` :

```
# mkdir -p /media/mmc
# mount /dev/mmcblk0p1 /media/mmc/
```

Depuis la console Linux, lancez le jeu :

```
# /media/mmc/armanoid &
```

...et voilà, admirez le résultat :

Figure10 : Le jeu sur PC

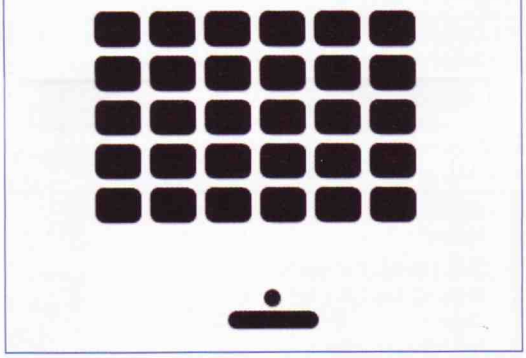
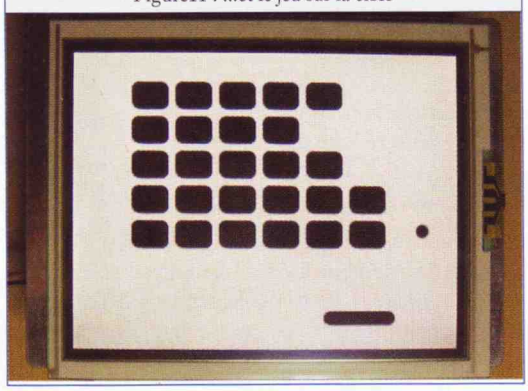


Figure11 : ...et le jeu sur la cible



Évidemment, sur une véritable console, il faudrait que, lors de l'insertion de la carte mémoire, un menu listant les jeux disponibles vous soit proposé et vous permette de les démarrer. Bien sûr, rien ne vous empêche de développer cette partie-là maintenant que vous possédez les bases ! Une autre évolution peut aussi consister à remplacer le clavier PS/2 par un keypad classique en utilisant le driver keypad du projet et à recompiler des jeux un peu plus funs, type Doom. Je vous invite à suivre dans les mois qui viennent ces évolutions dans la partie Multimédia du wiki.

3. Domaines privilégiés d'utilisation du projet

Même si l'exemple donné est orienté « ludique », de nombreux développements peuvent être envisagés autour du projet. Nous proposons un certain nombre de domaines dans lesquels nous souhaitons nous investir tout particulièrement :

- la domotique (gestion d'énergie, simulation de présence...) ;
- l'instrumentation (analyseur logique, oscilloscope, acquisition de données...) ;
- le multimédia : petites consoles de jeux, *player* MP3 réseau ;
- la robotique : commande et contrôle de robot.

Les nouvelles cartes d'extensions seront développées en priorité pour ces différents domaines.

Conclusion

Comme nous l'avons vu, le projet permet de développer relativement facilement et rapidement des systèmes complexes au premier abord. Son orientation *open source* permet aussi à chacun de participer en apportant ses idées et de se voir proposer de l'aide quant à leur réalisation. N'hésitez donc pas à venir nous rejoindre si vous avez envie d'approfondir vos connaissances en embarqué et êtes, comme nous, passionnés de Logiciels libres !

Julien Boibessot,

qui remercie Nicolas, Frédéric et Eric pour leur relecture,

julien.boibessot@armadeus.org

RÉRÉRENCES

- [1] La page de présentation de l'i.MXl chez Freescale : http://www.freescale.com/webapp/sps/site/prod_summary.jsp?nodeId=01 62468rH311432973ZrDR&code=i.MXL
- [2] Le WebPACK Xilinx pour le développement FPGA : http://www.xilinx.com/ise/logic_design_prod/webpack.htm
- [3] U-Boot, l'un des bootloaders les plus utilisés dans l'embarqué : http://u-boot.sourceforge.net
- [4] Le système de compilation de rootfs embarqué Buildroot : http://buildroot.uclibc.org
- [5] BusyBox, le couteau suisse de l'embarqué : http://www.busybox.net
- [6] Le site de la bibliothèque C embarquée uClibc : http://uclibc.org
- [7] Le toolkit pour l'embarqué Qtopia (ex Qt/Embedded) : http://www.trolltech.com/products/qtopia
- [8] La bibliothèque graphique SDL : http://www.libsdl.org
- [9] Le wiki de l'association : http://www.armadeus.org
- [10] Le projet sur SourceForge.net : http://sourceforge.net/projects/armadeus
- [11] Trivial File Transfer Protocol : http://fr.wikipedia.org/wiki/TFTP
- [12] C-Kermit : http://www.columbia.edu/kermit/ck80.html
- [13] La section « Matériel » du wiki où sont expliquées toutes les connexions physiques avec les matériels supportés par la carte : http://www.armadeus.com/wiki/index.php?title=Hardware
- [14] Pulse Width Modulation (Modulation de Largeur d'Impulsion) : http://fr.wikipedia.org/wiki/Modulation_de_largeur_d'impulsion
- [15] Formater correctement une MMC : http://www.armadeus.com/wiki/index.php?title=Fr:MultiMediaCard

Damien Krotkine

Introduction au framework Mason

Mason est un framework, basé sur mod_perl, qui permet de créer des sites web dynamiques. Puissant et polyvalent, son succès est grandissant. Utilisable aussi bien de manière autonome qu'encapsulé dans un autre framework, son terrain de jeu est très vaste.

Cet article présente la mise en œuvre de Mason dans un environnement classique Apache/mod_perl. Il sera suivi d'un second article traitant les concepts avancés, et, enfin, d'une étude de cas détaillée, avec code source complet.

1. HTML::Mason : Introduction

Perl Mason est un *framework* complet qui permet de « créer, servir et gérer des sites web ». Créé en 1998 et développé principalement par Jonathan Swartz et Dave Rolsky, Mason est plutôt facile à apprendre et polyvalent. Il peut se comparer à PHP pour sa facilité d'utilisation, à JSP pour son design moderne. Et surtout, Mason permet d'utiliser toute la puissance de Perl depuis un environnement totalement orienté web.

Une définition simple de Mason peut être : « un ensemble de modules pour dynamiser du contenu statique ». Cette description est très vague, car Mason peut générer n'importe quel type de contenu, même si on l'utilise principalement pour du HTML. Mason peut être appelé depuis un programme Perl autonome, ou bien à partir d'un CGI, voire être appelé depuis un framework de complexité plus grande, pour répondre aux besoins de mise en forme. Ainsi, Mason est une option parmi les différents modules de *template* de Catalyst, un MVC en Perl. Jifty[1] est un autre exemple de framework web prometteur basé sur Mason.

[1] Jifty est un framework web MVC basé sur Mason, développé principalement par Jesse Vincent. Même s'il est encore en cours de développement, les versions actuelles sont assez avancées pour être utilisées en production.

> MVC (Modèle Vue Contrôleur)
>
> C'est une méthode de conception d'applications, plutôt ancienne (1979) qui a été mise au goût du jour par le succès de frameworks web développés suivant ce principe. Cette architecture instaure une séparation entre les données (Modèles), la présentation (Vues), et le traitement des événements (Contrôleurs).

Mason est utilisé principalement dans le cadre de programmation web orientée page. Il est généralement mis en œuvre par-dessus le couple Apache/mod_perl. Ses atouts sont une syntaxe puissante, un jeu de modules proposant beaucoup de fonctionnalités, et la possibilité d'utiliser facilement des modules Perl tiers. Il peut également représenter une page en utilisant le modèle objet. Mason peut être vu comme le PHP de Perl, mais ne se limite pas à cela.

1.1 Hello World !

Commençons par l'inévitable *hello world*. Voici une page HTML enrichie grâce à la syntaxe de Mason :

```
<html>
<body>
% my $variable = 'World';
Hello <% $variable %>!
<%perl>
use POSIX qw(uname);
my $systeme = (uname())[0];
</%perl>
Ce serveur web utilise <% $systeme %>.
</body>
</html>
```

Après un examen rapide de ces quelques lignes, on peut en déduire les règles de base :

- Le texte simple (ou HTML, XHTML, etc.) sera transmis tel quel en réponse.
- Le caractère `%` en début de ligne permet d'évaluer du Perl facilement. Le reste de la ligne est considéré comme du Perl. Toutes les lignes de Perl d'un même fichier sont considérées comme étant dans la même portée lexicale. Attention, le caractère pourcent `%` doit vraiment être en début de ligne, il ne peut pas être précédé d'espaces.
- Lorsque le code Perl à exécuter dépasse une ligne, il est plus avantageux d'utiliser `<%perl>` et `</%perl>`. Ces balises permettent d'encapsuler des portions de Perl sur plusieurs lignes.
- La balise `<% %>` renvoie le résultat de l'évaluation du code Perl qu'elle contient. Elle est généralement utilisée pour afficher du contenu qui a été calculé précédemment dans une portion de code entre `<%perl>` et `</%perl>`, ou issu d'une ligne commençant par `%`.

Cet exemple n'est qu'un petit extrait de ce qu'il est possible de faire avec Mason, mais illustre la philosophie globale du framework : être accessible et facile d'utilisation, tout en restant concis. Ceci est possible grâce à l'utilisation de Perl directement dans la page web, contrairement à d'autres syntaxes, par exemple `Template::Toolkit`, qui nécessitent l'apprentissage d'un nouveau langage.

1.2 Exemples de syntaxe

Voici quelques structures de code qu'on rencontre couramment dans une page Mason :

1.2.1 Commentaires

Voici les différents types de commentaires :

```
%# ceci est un commentaire
<% # Ceci est également un commentaire, rarement utilisé %>
```

Attention, la première syntaxe requiert que le signe `%` soit le premier caractère de la ligne.

1.2.2 Condition

Afficher du contenu HTML de manière conditionnelle :

```
% if ($produit->get_stock() <= 0) {
  <span class='alert'>Produit indisponible</span>
% }
```

Cela permet de mélanger perl et HTML. Dans ce cas, la page affiche « Produit indisponible » uniquement si le produit n'est plus en stock.

1.2.3 Boucle

Au même titre qu'un test conditionnel, il est facile d'utiliser `for`, `foreach`, `while` et autres `<unless` :>

```
<ul>
% foreach my $attribut ($produit->get_attributs) {
%   my $cle = $attribut->{cle}
%   my $valeur = $attribut->{valeur}
    <li> <% $cle %> : <% $valeur %> </li>
% }
</ul>
```

1.2.4 Appel à un module Perl

Appeler un module Perl dans une page web est très simple. Voici un exemple de code qui vérifie si le drapeau `utf8` est vrai pour une variable :

```
<%perl>
  use Encode;
  my $chaine = "Exemple de chaîne de caractère avec accents";
  my $drapeau_utf8 = Encode::is_utf8();
</%perl>
% if ($drapeau_utf8) {
  <span class='info'>La représentation interne de la chaîne "<%
$chaine %>" est en UTF8</span>
% } else {
  <span class='info'>La représentation interne de la chaîne "<%
$chaine %>" est en latin1</span>
% }
```

Un autre exemple typique est d'utiliser `Data::Dumper` pour afficher le contenu d'une variable. Dans le code suivant, nous appelons la méthode `Dumper()` sur une variable `$a`, mais, au passage, nous remplaçons les retours à la ligne par des retours chariot HTML.

```
% use Data::Dumper;
% my $a = { foo => 'bar' }
<% join("<br>\n", split("\n", Dumper(\$a))) %>
```

1.3 Les différents intervenants

Un site Web basé sur Mason repose en général sur trois types de fichiers :

- les pages HTML + Mason ;

Il s'agit de pages de HTML (ou XHTML), pouvant contenir des balises Mason. Ces balises doivent être utilisées pour la forme (afficher un résultat de requête, du contenu), mais pas pour le fond (le calcul d'un résultat par exemple). Ces pages peuvent également être du HTML pur. On stocke généralement ces pages dans des fichiers en suffixe `.html`.

- les pages Mason pures ;

Ces pages ne contiennent que des instructions Mason. Comme nous le verrons plus tard, il est possible de créer des composants et des méthodes. Il arrive qu'on ait besoin d'effectuer des tâches dans l'environnement Mason, en utilisant les fonctionnalités des objets « request » de mod_perl `$r` et Mason `$m` (nous verrons ces deux objets plus bas). Ces pages ne doivent pas générer de contenu HTML sur la sortie standard, mais uniquement une valeur de retour. Ces fichiers ont souvent pour suffixe `.mhtml` ou `.mason`.

- les modules Perl.

L'intelligence du site doit être placée dans des modules Perl, qui seront appelés depuis les pages HTML ou les pages Mason. L'entrée et la sortie de ces modules doivent être bien définies. Ils ne doivent pas créer de HTML directement, mais générer des structures de données, qui seront mises en forme plus tard. Tout code qui n'a pas besoin d'utiliser l'environnement Mason ou mod_perl doit être placé dans un module Perl, qui, comme le veut la convention, est un fichier qui se termine par `.pm`.

Séparer les rôles de chaque page par type permet un développement plus structuré. Bien sûr, ce n'est qu'une convention, il faut donc avoir la volonté de la suivre. Cependant cela fait partie des *Bonnes Pratiques* du développeur Mason.

2. Configuration d'Apache

2.1 Mise en œuvre express

Mason est un module Perl, dont le nom exact est `HTML::Mason`[2]. Il peut être utilisé en mode autonome, en CGI, ou via mod_perl. Nous nous concentrerons sur l'utilisation de Mason avec le couple Apache/mod_perl.

Pour commencer, il nous faut un Apache avec mod_perl. Les exemples présentés ici utilisent Apache 1.x et mod_perl 1.x. Mason est compatible mod_perl 2 à partir de la version 1.29_02. Bien sûr, cela ne change rien à son utilisation. La plupart des distributions Linux permettent d'installer facilement le couple Apache/mod_perl. Une majorité d'entre elles proposent des paquetages Mason récents, mais si ce n'est pas le cas, il peut être installé à partir de CPAN.

Voyons rapidement la configuration du serveur Web. Ces lignes sont à rajouter dans le fichier de configuration d'Apache, par exemple `/etc/apache/httpd.conf` :

```
% use Data::Dumper;
% my $a = { foo => 'bar' }
<% join("<br>\n", split("\n", Dumper(\$a))) %>
PerlModule HTML::Mason::ApacheHandler

<Location />
  SetHandler   perl-script
PerlModule HTML::Mason::ApacheHandler

<Location />
  SetHandler   perl-script
  PerlHandler  HTML::Mason::ApacheHandler
  PerlSetVar   MasonDataDir /var/cache/apache
</Location>
```

Ceci permet de faire passer toutes les pages du site Web par l'interpréteur Mason. Celui-ci va analyser le

❷ L'auteur de Mason a indiqué qu'il avait baptisé le module HTML::Mason pour suivre les conseils de nommage à l'époque, mais que, s'il avait pu, il l'aurait simplement appelé Mason.

contenu, exécuter les commandes spéciales et produire une page résultat. On notera également la ligne

```
PerlSetVar MasonDataDir /var/cache/apache
```

« Code Cache »

Pour des raisons de performances, Mason précompile les pages à servir. Les objets précompilés sont stockés sur le disque dur, dans ce qu'on appelle le « Code Cache ». Cela n'a rien à voir avec le système de cache utilisable par le développeur, que nous verrons dans le prochain article.

Elle permet d'indiquer à Mason où stocker le contenu du « Code Cache ». Nous la spécifions ici, car c'est généralement la première chose qui bloque l'installation rapide de Mason, et la création d'une première page. En effet, ce répertoire doit être accessible en *écriture* par l'utilisateur système à qui appartient le processus Apache.

Ainsi configuré, Apache doit pouvoir servir des pages web Mason. Généralement, le serveur web affiche une page par défaut pour signifier qu'il fonctionne bien, située par exemple dans `/var/www/localhost/htdocs/index.html`. Nous pouvons tester si Mason fonctionne bien en introduisant une de ses balises dans ce fichier. `<% 6 * 7 %>` fera l'affaire. Si 42 apparaît lorsque la page est chargée, alors cela fonctionne correctement.

2.2 Configuration avancée

La configuration d'Apache que nous avons utilisée est un peu simpliste, voire franchement mauvaise, car tous les types de contenu seront interprétés. Cela pose un problème, car les images et autres fichiers binaires seront analysés. La performance s'en trouvera dégradée, et, pour peu que leur interprétation en ASCII contienne des balises Mason, le site web sera mal rendu. Filtrons donc l'application de l'analyseur syntaxique sur les suffixes qui nous intéressent :

```
PerlModule HTML::Mason::ApacheHandler

<LocationMatch "\.html$">
  SetHandler perl-script
  PerlSetVar MasonDataDir /var/cache/apache
  PerlHandler HTML::Mason::ApacheHandler
</LocationMatch>

<LocationMatch "(\.mhtml|dhandler|autohandler)$">
  SetHandler perl-script
  PerlInitHandler Apache::Constants::NOT_FOUND
</LocationMatch>
```

Nous indiquons à Apache qu'il doit servir les pages au suffixe `.html` via Mason, et renvoyer un 404 pour les `.mhtml`, mais également pour les *dhandler* et *autohandler* (voir plus bas), car nous ne voulons pas qu'un internaute puisse voir leur contenu.

Quand faut-il redémarrer Apache ? Mason est intelligent, il précompile les pages à servir, et se rend compte lorsqu'elles ont été modifiées. Ainsi, il n'est pas nécessaire de redémarrer Apache lorsque l'on modifie une page HTML + Mason ou Mason pure. Par contre, si votre site web utilise des modules Perl, il faudra redémarrer Apache à chaque changement dans ces modules (ou utiliser `Apache::Reload`).

2.3 Paramètres de configurations

Paramétrer Mason peut se faire depuis le fichier de configuration d'Apache, ou bien depuis du code Perl. Une règle de nommage est à connaître : en Perl, la notation est l'utilisation de minuscules et d'espaces soulignées (*underscore*, _). Quant au paramètre Apache, il s'écrit avec le préfixe Mason et avec des majuscules au début de chaque mot. Ainsi, on aura `parametre_en_perl` et `MasonParametreApache`.

Voici un exemple d'utilisation dans `httpd.conf` :

```
PerlSetVar MasonSessionClass Apache::Session::File
```

Cette ligne spécifie la classe à utiliser pour stocker les sessions.

Quelques paramètres souvent utilisés :

- `allow_globals` (`MasonAllowGlobals`) permet de spécifier une liste de variables globales à tous les composants.
- `comp_root` (`MasonCompRoot`) permet de spécifier la racine de la structure des composants, et comment les chemins des composants sont traduits en chemins de fichiers. Dans l'utilisation classique de Mason avec Apache ou en CGI, la valeur par défaut de `comp_root` est celle du `DocumentRoot`. Cependant, vous pouvez le changer. Vérifiez bien que l'utilisateur sous lequel est lancé Apache a bien accès à ce répertoire. Il est également possible de spécifier plusieurs racines.
- `data_dir` (`MasonDataDir`) est un paramètre important, qu'il est généralement nécessaire de configurer. Il spécifie l'endroit où Mason va stocker diverses données utilisées pour l'optimisation de l'exécution. L'utilisateur sous lequel Apache est lancé doit y avoir accès en *lecture/écriture*. Le contenu de ce répertoire peut être effacé, Mason ne l'utilise que pour des questions de performances. Sa valeur par défaut est dépendante de votre distribution.

2.4 Tester la configuration

Une fois configuré, il ne reste plus qu'à démarrer le serveur Apache, et placer un fichier de test à la racine du site web. Par exemple, `text.html` qui contient :

```
<html><body>
Test : <% 40 + 2 %>
</body></html>
```

En ouvrant un navigateur web sur cette page, le nombre 42 devrait apparaître.

3. Les composants Mason

3.1 Description

Un composant est un morceau de site web. Un site web est généralement constitué de plusieurs pages. Ces pages sont construites à partir de plusieurs éléments : par exemple, un menu, un bandeau supérieur, un bandeau inférieur, un cadre de contenu, un encart de navigation, etc. Les composants Mason permettent d'implémenter ces éléments, et de les assembler. Prenons en exemple un petit site personnel. Considérons que chaque page est structurée comme suit :

- un bandeau supérieur, contenant un logo et la date du jour ;
- un menu de navigation vers les autres pages ;

- le contenu de la page proprement dit ;
- un bandeau inférieur, contenant des mentions légales.

Ce site web contient plusieurs pages :

- une page d'accueil ;
- une page de brèves ;
- une page « à propos de l'auteur ».

Nous pouvons structurer le site web comme suit : un composant pour chaque page, lui-même constitué de composants pour les bandeaux supérieur et inférieur, la navigation et le contenu.

3.2 Syntaxe

Un composant est un fichier. Il n'a pas besoin de déclaration. Un fichier `test.html` contenant du HTML et du Mason est un composant sans le savoir.

Lors d'une requête sur le site web, Mason doit décider quel composant (appelé « *top-level* ») va être interprété en premier. Il est choisi en fonction de l'URL. Par exemple, de l'adresse **http://www.siteweb.com/produits/cuillere.html?modele=7**, Mason déduit qu'il lui faut appeler le composant `cuillere.html`, dans le répertoire `produits/`. S'il n'existe pas (si le fichier `/produits/cuillere.html` est inconnu), alors Apache renverra une erreur 404. En fait, nous verrons plus tard qu'il est possible d'intercepter ce 404.

3.2.1 Appeler un composant

Nous venons de le voir, un composant peut s'appeler simplement en entrant son URL. Mais les composants peuvent bien sûr s'appeler entre eux. On utilise cette syntaxe :

```
<& chemin/du/composant.html, arguments_optionels &>
```

Le composant appelé sera interprété, et le contenu HTML résultant inséré en lieu et place de l'appel. Le chemin du composant suit la syntaxe des URI, donc le séparateur est `/`, et il peut être absolu ou relatif. Le répertoire courant est celui du composant appelant.

Voici le composant de la page d'accueil. Nous allons le créer dans `index.html`. Ce sera la page par défaut de notre site web d'exemple :

```
%# index.html
%# Début du composant index
%# on appelle le bandeau supérieur
  <& bandeau_sup.html &>
%# on appelle le menu de navigation
%# on passe le nom de la page actuelle
  <& navigation.html, page_courante => 'index' &>
%# Voici le contenu spécifique de la page d'accueil
<center>Bonjour et bienvenue sur ce site web !</center>
%# le bandeau inférieur
<& bandeau_inf.html &>
```

Créons maintenant le composant du bandeau supérieur qui s'appelle `bandeau_sup`, qui va contenir les balises de début de page, et un logo. Éditons donc le fichier `bandeau_sup.html` :

```
%# bandeau_sup.html
<html>
<head></head>
<body>
<center><img src="/images/logo.png"></center>
```

C'est plutôt simple. Bien sûr, ce HTML n'est qu'un exemple, il n'est pas du tout conforme aux normes W3C. Voyons maintenant le composant de navigation. Le menu de navigation doit présenter un lien vers les autres pages du site web, mais pas vers celle affichée actuellement, car c'est redondant. Ainsi, sur la page d'accueil, le menu de navigation permet de se rendre sur la « page de brèves », ou vers l'« à-propos ». Sur la page « à-propos », le menu de navigation propose les liens vers la « page d'accueil » ou la « page de brèves». C'est pourquoi nous passons en argument le nom du lien à ne pas afficher (voir plus bas en détail le passage d'arguments). Voici le composant navigation :

```
%# navigation.html
%# récupération de l'argument
<%args>
$page_courante
</%args>
%# on commence une table
<table>
%# on boucle sur tous les liens
% foreach(qw(index breves apropos)) {
%# si le lien n'est pas la $page_courante, on l'affiche
%   if ($_ ne $page_courante) {
%# on affiche le HTML du lien, construit à la volée, dans une ligne de la table
      <tr><td> <a href="<% $_ . 'html' %>">$_</a> </td></tr>
%   }
% }
%# on ferme la table
 </table>
```

Et pour finir, voici le source du bandeau inférieur qui ferme la page web :

```
<& chemin/du/composant.html, arguments_optionels &>
```

Le fichier `index.html` est chargé par défaut lorsque l'utilisateur navigue sur le site web. Il est cependant également possible d'appeler les autres pages (donc composants) du site. Ainsi l'URL **http://www.siteweb.com/bandeau_sup.html** permet d'appeler le composant `bandeau_sup.html`.

3.2.2 Passer des arguments

Voici quelques précisions sur le passage d'arguments d'un composant à un autre. Les arguments peuvent être de tout type : scalaire (nombre ou chaîne de caractères), tableau ou *hash*. Ils peuvent être obligatoires ou optionnels. Voici un petit code d'exemple de passage d'arguments. Le prototypage est fait dans le composant appelé :

```
<%args>
$a
@b      # des arguments obligatoires, de type scalaire, tableau ou hash
%c
$d => 5  # voici un argument scalaire optionnel, et sa valeur par défaut
$e => $d * 2
@f => ('foo', 'bar')
%g => (joe => 1, bob => 2)
$h => undef # optionnel, par défaut undef
</%args>
```

Et @_ ?

Si vous n'aimez pas le passage de paramètres nommés, il est toujours possible d'utiliser @_ :

```
# côté appelant
<& composant, 42, 'chaîne' &>
# côté appelé
my ($reponse, $chaine) = @_;
```

Le passage des arguments se fait par nom. $a, @b et %c sont des arguments obligatoires, de type scalaire, tableau et hash. $d est un argument scalaire optionnel, car il a une valeur par défaut définie, 5. $e est également optionnel, mais sa valeur par défaut est dépendante de $d. @f et %g sont des arguments optionnels, avec des valeurs par défaut. La définition des arguments est à mettre entre les balises <%args> et </%args>. Les variables ainsi définies peuvent être utilisées dans le reste du composant, et seront locales au composant. La règle à retenir est : un argument avec une valeur par défaut est optionnel, sinon il est obligatoire.

Du côté du composant appelant, voici comment passer des arguments :

```
% <& composant, $a => 5 &>
% <& composant, @b => ['une', 'référence', 'de', 'liste'] &>
% <& composant, %c => { un => 'hash', en => 'référence'} &>
```

Il est possible de faire appel à des fonctions avancées de passage d'arguments, comme utiliser @_ ou %ARGS directement, mais nous ne les aborderons pas ici, car elles sont plus rarement utilisées.

Nous avons vu précédemment qu'il est possible d'accéder à un composant directement par son URL. Bien sûr, il est possible de spécifier des arguments dans l'URL. Ainsi, pour appeler le composant navigation.html en lui passant un nom de page, nous utiliserons cette syntaxe :

```
http://www.siteweb.com/navigation?page_courante=index
```

Voici comment passer en argument d'URL le tableau @b et le hash %c, vu ci-dessus :

```
http://www.siteweb.com/composant?b=une&b=reference&b=de&b=liste
http://www.siteweb.com/composant?c=un&c=hash&c=en&b=reference
```

Params::Validate

Ce style de passage d'argument est disponible en dehors de Mason grâce au module Params::Validate de Dave Rolsky (un des auteurs de Mason). Ce module permet de passer des arguments comme le fait Mason, et de les vérifier.

Le passage d'arguments dans Mason est simple et puissant, et couvre tout type de donnée. Comme dans la quasi-totalité des frameworks web modernes, il est également transparent, c'est-à-dire que le développeur web n'a pas à se soucier de la provenance des arguments, ni de leur récupération manuelle. Ainsi, comme il est facile de passer des arguments d'un composant à un autre, il est possible (et recommandé) de faire beaucoup de composants distincts, un par fonctionnalité.

3.3 Les sections spéciales

3.3.1 Définition

Il existe quelques balises spéciales en Mason, qui permettent de déclarer des sections à l'intérieur d'un composant. Les sections sont des blocs qui contiennent du Perl uniquement, qui sont donc exécutées dans un contexte particulier.

3.3.2 <%init>

Commençons par une section simple : <%init>. Le contenu de ce bloc sera interprété au début du composant, même si la section <%init> est définie à la fin du composant. C'est utile pour séparer l'initialisation de variables Perl, qu'on peut reléguer à la fin du composant, et le contenu HTML de la page, qu'il est préférable de présenter au début du composant. Voici un exemple de syntaxe de ce bloc :

```
%# début du composant
 <html>
   <head>
     <title>titre</title>
   </head>
 <body>
 %# Contenu de la page, qui utilise des variables
 Bonjour, nous sommes le <% $day %> du mois.
 </body>
 </html>
 %# Section init, utilisée pour initialiser $day
 <%init>
   $day = (localtime(time))[3];
 </%init>
```

Dans cet exemple, la section <%init> évite d'avoir à écrire la ligne d'initialisation de $day au tout début du composant. On peut placer <%init> n'importe où dans le composant, on est sûr que son code sera interprété au début du composant. Les lecteurs avertis se demanderont s'il est possible d'avoir plusieurs sections <%init> dans un seul composant. La réponse est : oui, c'est possible, même si ce n'est pas conseillé. Dans le cas de plusieurs sections <%init>, le code sera interprété dans l'ordre des sections <%init>.
Il est utile de noter que <%init> est interprété à chaque appel du composant, contrairement à d'autres sections.

3.3.3 <%cleanup>

<%cleanup> est l'inverse de <%init>, son contenu est exécuté à la fin du composant. <%cleanup> est rarement utilisé, car les variables normales du composant sont détruites à la fin de celui-ci (on dit qu'elles sortent de la portée du composant). <%cleanup> est donc utile uniquement dans le cas où un traitement spécial est à effectuer à la fin, par exemple libérer une connexion à une base de données.

3.3.4 <%once>

<%once> est une section dont le contenu est exécuté une seule fois, à l'initialisation du composant. Apache crée plusieurs processus (ou des *threads*, selon sa configuration), pour répondre aux requêtes des utilisateurs du site web. Un composant est initialisé la première fois qu'il est demandé par un utilisateur,

pour un processus donné. Une fois initialisé, le composant est précompilé, et mis en cache interne de Mason. `<%once>` permet d'exécuter du code Perl lors de cette initialisation. `<%once>` ne sera plus jamais appelé avant qu'un nouveau processus soit créé par Apache, ou que Apache soit redémarré.

Quelle est donc l'utilité de `<%once>` ? Cette section est très pratique pour déclarer des variables, qui seront valides pendant toute la durée de vie du processus, et qu'on pourra initialiser une seule fois. Par exemple, il est pratique d'utiliser conjointement `<%init>` et `<%once>` pour créer une connexion à une base de données permanente :

```
<%once>
  # déclaration de l'objet database handler
  use DBI;
  my $db;
</%once>

%# Au début de l'appel du composant,
%# on initialise $db si besoin

<%init>
  if (!defined $db) {
    $db = DBI->connect('dbi:mysql:dbname=foobar');
  }
</%init>

%# $db est utilisable dans le composant
```

3.3.5 <%filter>

Dans une section `<%filter>`, la variable spéciale `$_` contient le HTML final de la page, tel qu'il va être renvoyé. `<%filter>` permet donc de modifier le résultat final d'un composant : il suffit d'altérer `$_`. L'exemple suivant passe le HTML de la page en majuscule.

```
<%filter>
y/a-z/A-Z/;
</%filter>
```

La section `<%filter>` est intéressante pour opérer une transformation sur le flux en sortie, sans se préoccuper de la manière dont il a été créé. Cependant, cela peut nuire à la lisibilité et à la maintenabilité du code, car le fait de multiplier les interventions sur les données, dans des endroits différents du code, peut complexifier les processus.

Autres sections

▶ `<%shared>`
Similaire à `<%once>`, mais les variables déclarées dans cette section ont leur portée limitée à la requête, et non au processus Apache.

▶ `<%def>`
Permet de définir un *sous-composant*. La convention veut que les noms des sous-composants commencent par un point.

▶ `<%method>`
Permet de définir une *méthode* du composant. Les méthodes seront expliquées dans le prochain article.

▶ `<%flags>` et `<%attr>`
Permettent d'assigner des états ou attributs aux composants. Utilisés principalement dans la programmation objet (voir le prochain article).

▶ `<%doc>`
Son contenu est traité comme des commentaires.

3.3.6 Exemple d'utilisation des sections

Voici un exemple concret d'utilisation des sections. Pour un développement web conséquent, il peut être important de soigner la forme du code HTML renvoyé. Notamment, un code HTML source bien indenté est utile pour repérer et comprendre les problèmes d'affichage d'une page web. Mais indenter du HTML dans un composant Mason n'est généralement pas possible : certaines portions d'HTML sont incorporées dans certains cas uniquement. On ne peut pas prévoir l'indentation avant que les pages ne soient générées totalement.

La solution est d'indenter le HTML à la volée, avant de le renvoyer vers le navigateur de l'internaute. Ce faisant, nous pouvons également en profiter pour analyser syntaxiquement le HTML grâce à un module ad hoc ; si cet analyseur détecte des erreurs, les lignes incriminées peuvent être affichées en bas de page, en couleur.

Après une petite recherche sur CPAN, `HTML::Tidy` paraît l'outil idéal pour vérifier la syntaxe d'un source HTML, et l'indenter. Tout d'abord, il faut organiser le site pour que tous les composants héritent d'un composant racine, que nous appelons `tidy.html`. Cela peut se faire à l'aide d'un autohandler (voir plus bas), ou bien grâce à l'orienté objet (expliqué dans le prochain article). Ce composant indente et analyse le contenu avant qu'il ne soit retourné à l'internaute. Voici le code commenté. Il illustre l'utilisation des balises `<%once>`, `<%init>` et `<%filter>`.

```
%# Nous allons utiliser un objet tidy valable
%# pour toute la durée de vie du processus alloué par Apache.
%# %once est idéal pour une telle déclaration
<%once>
  # déclaration de l'objet d'indentation du code
  use HTML::Tidy;
  my $tidy;
</%once>

%# Au début de l'appel du composant, il nous faut
%# affecter $tidy, s'il n'est pas déjà défini.
<%init>
  # initialisation si besoin est, pour être fork-safe
  if (!defined $tidy) {
    $tidy = HTML::Tidy->new( { config_file => '/tmp/tidy.cfg' } );
  }
</%init>

%# On applique un filtre pour auto-indenter le html qui sort.
%# Pour cela, on utilise la section %filter, qui est appelée en
%# fin de calcul du composant.
<%filter>
  # récupération du HTML
  my $html = $_;
  # on nettoie la sortie du composant
  $html = $tidy->clean($html);

  # s'il y a des erreurs ou des avertissements,
  # on les affiche à la fin
  if ($tidy->messages) {

        # récupération des erreurs ou avertissements
      my @messages = $tidy->messages();

      # On ajoute les erreurs et avertissements
      # HTML à la fin du contenu
        # si le _type du message est à 1,
      # c'est un warning, sinon une erreur
      $html .= '<br><br><hr size=2 '.
```

```
            'noshade>html warnings/errors : <br>' .
            join("\n<br>", map  {
                ($_->{_type} ? 'warning : ' : 'error : ')
                        . $_->as_string;
            } ($tidy->messages)) . '<br><br>';
    }
    $_ = $html;
    # on réinitialise l'objet tidy
    $tidy->clear_messages();
</%filter>
```

4. $r et $m : deux couteaux suisses

Il est temps de présenter rapidement les deux objets principaux accessibles dans l'environnement Mason : `$r` et `$m`. Ces deux objets sont globaux, et utilisables partout sans avoir besoin de les déclarer. En réalité, il existe des exceptions, mais elles sont très rares. On peut considérer qu'on a accès à `$r` et `$m` dans toute situation.

4.1 $r

`$r` est l'objet `request` de mod_perl. Il permet un accès à l'API Perl d'Apache, ce qui est la grande force de mod_perl. Ainsi, on pourra utiliser :

```
$r->uri;            # L'URI HTTP (équivalent à l'URL dans bien des cas)
$r->header_in(..);  # permet de récupérer les entêtes HTTP
$r->content_type;   # permet de récupérer ou de changer le content-type
$r->header_out(..); # permet de récupérer ou de changer un entête en sortie
```

4.2 $m

`$m` est un accès à l'API Mason. Ainsi, `$m` offre un accès au mécanisme de recherche et d'appel aux composants, permet d'accéder aux paramètres en cours et de les modifier. On peut même créer une requête Mason de toutes pièces, et déclencher un appel de composant dessus.

`$m` est une instance de la classe `HTML::Mason::Request`, et son API est conséquente. Une partie est la plus souvent utilisée, il s'agit de `HTML::Mason::Component`, qui concerne les composants. Comment travailler sur un composant ? `$m` permet d'obtenir une instance de `HTML::Mason::Component` facilement :

▶ `$m->current_comp`

Renvoie le composant courant.

▶ `$m->callers`

Sans argument, renvoie la liste de la pile des composants parents. On peut lui passer un nombre pour récupérer un composant parent précis.

▶ `$m->fetch_comp(chemin)`

Renvoie le composant correspondant au chemin.

Une fois le composant récupéré, il est possible de travailler avec. On peut récupérer son chemin, ses attributs, ses drapeaux. On peut également savoir si une méthode existe et l'appeler, etc.

Nous allons voir rapidement comment utiliser certaines méthodes de `$m` dans les chapitres sur les dhandlers et les autohandlers.

5. Les handlers

5.1 Le dhandler

Dans le chapitre sur « Les composants Mason », nous avons expliqué rapidement comment était choisi le composant « top-level ». Le principe de base est simple : à un composant est associé un fichier du même nom. Cependant, que se passe-t-il si le fichier composant n'existe pas ? Avant de renvoyer une erreur, Mason parcourt le répertoire du composant et ses répertoires parents à la recherche d'un fichier spécial appelé `dhandler`. Dhandler veut dire « *default handler* », « gestionnaire par défaut ».

Dès qu'un dhandler est trouvé, Mason l'exécute et arrête sa recherche. Un dhandler est un composant comme un autre, sauf qu'il peut utiliser à son avantage `$m->dhandler_arg`. Cette valeur de l'objet Mason (`$m`), contient le chemin vers le composant « top-level » qui n'existait pas.

Reprenons l'exemple donné dans le chapitre sur les composants, avec l'URL

```
http://www.siteweb.com/produits/cuillere.html?modele=7
```

Nous pouvons créer un dhandler pour gérer le cas d'une demande d'un produit qui n'existe pas, et afficher un message d'erreur plus humain qu'un 404. Pour cela, nous allons créer un dhandler dans le répertoire produit, qui affichera simplement que le produit n'existe pas. Voici le code de `/produit/dhandler` :

```
<html>
<body>
<b>Il n'y a pas de <% $produit %>.</b>
</body>
</html>

<%init>
# on récupère le chemin complet (absolu) du produit demandé
my $arg = $m->dhandler_arg;
# on prend le dernier élément du chemin,
# donc le nom du composant demandé
my $produit = (split('/', $arg))[-1];
# on enlève le suffixe .html, ce qui donne le nom du produit
$produit =~ s/\.html$//;
</%init>
```

Avec ce dhandler, lors de la requête, Mason renverra :

```
Il n'y a pas de cuillère.
```

Le dhandler peut donc être utilisé comme reprise sur erreur, mais nous allons voir qu'il est très intéressant de l'utiliser pour appeler le contenu adéquat. Il suffit de choisir une convention dans la syntaxe des URL, et d'utiliser le dhandler pour interpréter l'URL comme un ordre. L'exemple classique est celui du ***blog***. Un blog est une collection de billets (petits textes) classés, qu'on peut atteindre par date ou par sujet par exemple. Les billets du blog sont stockés dans une base de données. Pour simplifier, on considérera que les billets sont stockés dans une unique table, avec une colonne pour la date et une pour le sujet du billet.

Nous pouvons convenir d'une convention d'URL comme suit :

```
http://www.siteweb.com/blog/par_date/20060701
http://www.siteweb.com/blog/par_sujet/informatique
```

Dans le répertoire `/blog`, nous allons écrire un dhandler qui interprétera l'URL suivant cette convention : le premier élément du chemin doit être `/blog`, le second indique sur quelle colonne de la table on filtre, et le troisième élément du chemin indique la valeur du filtre. Voici à quoi le dhandler ressemblera :

```
<html>
<body>
<% $contenu %>
</body>
</html>
<%init>
# on considère qu'une connexion $DBH à
# la base de données a été faite
# on récupère le chemin
my $arg = $m->dhandler_arg;
# on en fait un tableau en séparant sur /
my @args = split('/', $arg);
# on vérifie que le premier élément du chemin est /blog
shift @args eq 'blog' or $m->decline;
# on prend les deux prochains éléments
my ($colonne, $valeur) = @args;
# on exécute la requête dans la base de donnée
my $sth = $DBH->prepare(
      qq{SELECT contenu FROM blog WHERE ? = ? } );
$sth->execute($colonne, $valeur);
my $lignes = $sth->fetchall_arrayref;
# S'il n'y a pas de billet correspondant, on renvoie un 404
@$lignes or return 404;
# à partir du résultat de la requête, on construit $contenu
my $contenu = ...
</%init>
```

Ce mécanisme permet d'accéder de manière souple et puissante au contenu du blog. Si un jour vous ajoutez une colonne dans la table blog, aucune modification n'est à faire sur le code Mason, pour pouvoir l'utiliser dans le site web.

Les lecteurs attentifs auront noté l'utilisation de `$m->decline`. Cette commande permet de dire que ce handler ne sait pas gérer la requête actuelle, et que Mason doit continuer à chercher un autre handler qui saura traiter la requête. Alors que `return 404` est définitif.

Pour conclure, le dhandler est très similaire à `AUTOLOAD` en Perl. Il permet d'intercepter une requête a priori mauvaise, et d'effectuer une action à la place.

5.2 Autohandler

Une fois le principe du dhandler compris, il n'est pas difficile de comprendre ce qu'est un autohandler. Son nom veut dire « *automatic handler* », « gestionnaire automatique », et il fonctionne par répertoire. Lorsque Mason a trouvé le composant « top-level », il vérifie juste avant de l'exécuter s'il existe un fichier appelé `autohandler` dans le répertoire. Si c'est le cas, il l'exécute à la place, avec une petite subtilité : le composant sera ajouté à la pile d'appel.

Le fichier `autohandler` est un composant classique, mis à part le moment où il est appelé. Prenons l'URL d'exemple

```
http://www.siteweb.com/produits/cuillere.html?modele=7
```

le composant top-level est `cuillere.html`, dans le répertoire `/produits`. S'il existe un fichier `/produits/autohandler`, il va être exécuté à la place de `cuillere.html`. Cependant, `cuillere.html` va être ajouté dans la pile d'appel de Mason. Ainsi, si l'autohandler fait appel à `$m->call_next`, alors le composant `cuillere` sera exécuté, puis Mason redonnera la main à l'autohandler.

L'intérêt immédiat est la possibilité d'encapsuler tous les composants d'un répertoire. Voici un autohandler trivial, qui ajoute un en-tête et un bas de page à tous les composants situés dans le répertoire :

```
<html>
<head><title>Mon site web fantastique</title></head>
<body>
% $m->call_next; # ceci appelle le composant originel
<hr />
Ceci est exemple de bas de page.
</body>
</html>
```

L'autohandler permet également de factoriser du code. Par exemple, sur un site web utilisant beaucoup de connexions à une base de données, chaque composant peut vouloir vérifier qu'il dispose d'une connexion à cette base, et la créer si besoin est. Au lieu d'écrire le code dans chaque composant, on peut le factoriser dans un autohandler.

Les autohandlers sont aussi utilisés pour inclure automatiquement des composants Mason, des modules Perl, ou même du javascript. Dans un site web utilisant beaucoup la bibliothèque javascript `Prototype`, il sera intéressant d'écrire un autohandler comme suit :

```
<html>
<head>
  <title>Titre du site web</title>
  <script src="../chemin/de/prototype-1.4.0.js"></script>
</head>
<body>
% $m->call_next; # ceci appelle le composant originel
</body>
</html>
```

En essayant d'élargir le concept, on peut utiliser une combinaison d'un dhandler et d'un autohandler, ce qui permet de déterminer un composant top-level en fonction de règles propres au site, et d'hériter du autohandler certaines actions. On peut également avoir plusieurs autohandlers, qui seront appelés successivement. Cependant, ce fonctionnement est assez difficile à gérer, et on atteint assez vite ses limitations.

Ces problèmes sont résolus par l'utilisation de la programmation orientée objet de Mason, que nous aborderons dans le prochain article.

Conclusion

C'est la fin de cet article, mais pas de l'aventure ! Nous venons de voir les bases de Mason, qui devraient vous permettre de vous lancer dans l'implémentation d'une première application Web dynamique. Dans le prochain article, nous aborderons entre autres les fonctionnalités Orientées Objets de Mason, l'utilisation du système de cache intégré, et nous testerons un module Ajax.

LIENS

- MasonHQ : http://www.masonhq.com
- Vue Mason pour Catalyst : http://search.cpan.org/~ank/Catalyst-View-Mason
- Jifty : http://jifty.org

Damien Krotkine,

<dams@zarb.org> – Paris.pm

H. Fernandes, S. Ducasse

Smalltalk et les design patterns, un couple assorti

Les design patterns sont des descriptions de solutions de conception qui créent un vocabulaire pour les développeurs. Smalltalk a une longue histoire d'amour avec les design patterns. A l'origine de l'émergence de la notion même de patterns, on retrouve deux développeurs Smalltalk célèbres, K. Beck et W. Cunningham. De plus, de nombreux design patterns ont été développés en Smalltalk avant même qu'ils soient reconnus comme design patterns. Avec cet article, nous abordons la présentation de trois design patterns importants dans la construction des interfaces graphiques : les patterns MVC, Observer et Command. Dans un prochain article, nous aborderons des patterns essentiels comme le Composite, Visitor, Singleton et autres.

Les design patterns

Les *design patterns* sont des descriptions de solutions *récurrentes* à des problèmes de conception *récurrents*. Ils sont très importants, car ils constituent un socle de vocabulaire qui permet aux développeurs de communiquer plus rapidement. Il est très important si vous lisez un livre sur les design patterns que vous ne preniez pas pour graver dans le marbre l'implémentation proposée. Il s'agit *d'une* solution possible. Le plus important dans un design pattern est son **nom,** son **intention** et la **solution**. Son nom crée le vocabulaire qui va permettre aux développeurs d'échanger et communiquer de manière plus efficace. Ensuite, la discussion de son applicabilité et les problèmes potentiels qu'il introduit sont aussi importants. Les patterns ne font pas des miracles et bien comprendre quand on ne doit pas appliquer un pattern est aussi important que le connaître. Nous montrerons dans un autre article le Singleton qui permet à une classe de s'assurer qu'une seule de ses instances est active a un moment donné. C'est le pattern le plus simple et le plus mal appliqué, même par les experts.

Les design patterns sont très intéressants, car ils permettent aux développeurs débutants d'avoir accès à des solutions *éculées, connues et validées* des experts. Notez les termes en italique dans la phrase précédente, ils sont capitaux : un pattern n'est pas une astuce technique ou un *hack* spécifique à un petit problème spécifique, mais une solution souvent simple a un problème qui se répète. On dit qu'un pattern existe s'il est apparu 3 fois dans des applications ou *frameworks* différents. Donc, méfiez-vous des nouveaux patterns qu'un expert local aurait découverts. Un pattern met aussi en jeu des forces : introduction de plus de classes, plus d'objets, complexification de la conception versus la flexibilité accrue.

Un brin d'histoire. Les design patterns et Smalltalk sont une longue histoire d'amour. En effet, Kent Beck le guru Smalltalk qui a inventé eXtreme Programming et remis à la mode les tests unitaires, a été un des premiers avec Ward Cunningham à créer un groupe de discussion autour des patterns : le *Hillside group*. Selon ses propres dires, Ralph Johnson, un des quatre auteurs du fameux livre *Design patterns*, aussi appelé *Gang of Four* (GOF), était arrivé aux premières réunions de travail avec tous les patterns écrits en Smalltalk. Par ailleurs, nous vous conseillons de lire le livre *The Design Pattern Smalltalk Companion* d'Alpert, Brown et Woolf, car il est plus lisible que l'original (qui est en phase de réécriture). Même certains programmeurs C++ et Java le préfèrent d'après Amazon. En effet, ce livre est apparu 4 années après le premier et a capitalisé sur la compréhension du processus d'écriture des design patterns.

Il est aussi important de ne pas se soucier si on n'implémente pas un pattern dans son application favorite. Nous le répétons : savoir *quand* appliquer ou pas un pattern est plus important que l'appliquer. De plus, tous les patterns écrits dans le GOF ne sont pas au même niveau d'importance. La version 2 du GOF verra une certaine refonte dans les patterns présentés. Notez qu'il existe aussi d'autres livres sur les patterns comme la série *Pattern Languages of Program Design*, mais, même parmi ces livres, certains patterns ne font pas sens. Donc, utilisez votre expérience et sens critique pour savoir si un pattern résout vos problèmes.

Dans cet article, nous présentons les patterns MVC/Observer et Command. Ils sont relativement importants car ils sont très souvent utilisés lors du développement des applications graphiques interactives. Pour illustrer ces patterns, nous prendrons comme prétexte l'écriture d'une petite application graphique interactive que nous appelons `Bob`. Il s'agit d'une entité que l'on déplace à l'écran et dont nous pouvons changer la couleur.

MVC et Observer

L'objectif du pattern MVC est de clarifier les différentes couches d'une application graphique interactive. Ce pattern fut inventé en Smalltalk vers la fin des années

70 par Tryve Reenskaug (qui participe à la *mailing-list* Squeak et travaille à BabyUML).

Pour un élément et sa représentation graphique interactive, trois niveaux sont distingués :

- le **modèle** qui représente l'objet logique manipulé ;
- la **vue** qui est une représentation visuelle de l'objet logique, c'est-à-dire du modèle ;
- le **contrôleur** qui permet d'interagir avec la représentation visuelle du modèle.

Dans le cas d'un paragraphe de texte, le modèle est un objet contenant le texte ; la vue est une représentation graphique du modèle sous forme de paragraphe formaté ; le contrôleur représente les éléments d'interaction avec le paragraphe de texte : menus et boutons spécifiques.

Ces trois couches sont modélisées par trois classes qui pourront porter ces mêmes noms. Une vue, pour être capable d'afficher le modèle, référence donc dans une variable d'instance le modèle. Il en est de même avec le contrôleur qui doit faire remonter vers le modèle les modifications faites par l'utilisateur. Lorsque le modèle a été modifié, celui-ci pourrait signaler les modifications à sa vue par l'intermédiaire d'une variable d'instance, mais ce n'est pas le cas. En fait, le modèle ne doit pas avoir connaissance de sa vue. Après tout, un modèle peut très bien avoir plusieurs représentations graphiques. C'est ici qu'intervient le pattern Observer. Celui-ci permet à des objets – des vues – d'être notifiés de changements dans un autre objet – un modèle. Le modèle émet des événements pour dire qu'il change, mais il ne sait pas à qui ses vues sont notifiées.

La vue et le contrôleur sont relativement proches, l'un référence l'autre par l'intermédiaire de variables d'instance. De plus, leur implémentation est très dépendante, à un type de représentation graphique donnée correspond un ensemble de mode d'interaction avec la représentation graphique. Dans certains cas, on peut fusionner la vue et le contrôleur, car ils ne peuvent pas varier indépendamment l'un de l'autre. C'est une des critiques de MVC, par ailleurs.

Commençons par écrire notre application **Bob**. Il est représenté par une ellipse. Nous cliquons dessus pour lui donner des ordres, et nous le déplaçons alors avec les touches fléchées. Nous pouvons également changer sa couleur. Les deux caractéristiques de Bob sont donc sa position à l'écran et sa couleur, ce sont donc deux attributs de son modèle :

```
Model subclass: #BobModel
        instanceVariableNames: 'color position'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'LinuxMag92'
```

Notre modèle est défini à partir d'une classe `Model`, préexistante dans Smalltalk. Nous définissons l'initialisation et les méthodes d'accès aux attributs :

```
BobModel>>initialize
        super initialize.
        color := Color cyan.
position := 200 atRandom @ 200 atRandom
BobModel>>color
        ^ color
BobModel>>position
        ^ position
```

Passons à la définition de la vue :

```
EllipseMorph subclass: #BobView
     instanceVariableNames: 'controller model domain'
     classVariableNames: ''
     poolDictionaries: ''
     category: 'LinuxMag92'
```

`domain` référence un objet qui représente notre application. C'est cet objet qui nous permet de la démarrer.

En regardant l'héritage, il est clair que notre Bob ressemblera à une ellipse. Définissons également notre contrôleur :

```
Controller subclass: #BobController
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'LinuxMag92'
```

Il est défini à partir de `Controller` dans lequel les variables d'instance `model` et `view` sont déclarées.

Étoffons notre vue avec quelques méthodes :

```
BobView>>initialize
        super initialize.
        controller := BobController new view: self.
        self borderWidth: 2
BobView>>handlesMouseDown: anEvent
        ^ true
BobView>>mouseDown: evt
"Prendre  ou laisser le focus clavier lorsqu'on me clique dessus"
        evt hand keyboardFocus == self
                ifTrue: [evt hand newKeyboardFocus: nil]
                ifFalse: [evt hand newKeyboardFocus: self]
BobView>>keyboardFocusChange: aBoolean
"changer mon aspect selon que j'ai gagné ou perdu le focus clavier"
        super keyboardFocusChange: aBoolean.
        aBoolean
                ifTrue: [self borderColor: Color red]
                ifFalse: [self borderColor: Color black]
```

Pour le fonctionnement des méthodes ci-dessous, vous pouvez vous reporter à l'article de *GNU Linux Magazine* 90. En quelques mots, pour donner des ordres clavier à Bob, nous devons d'abord cliquer dessus. Lorsque Bob est prêt à recevoir de tels ordres, son contour devient rouge, sinon il est noir.

```
BobView>>handleKeystroke: evt
        | keyValue |
        keyValue := evt keyValue.
        keyValue = 30
```

```
                ifTrue: [controller up].
        keyValue = 31
                ifTrue: [controller down].
        keyValue = 29
                ifTrue: [controller right].
        keyValue = 28
                ifTrue: [controller left].
        keyValue = 99
                ifTrue: [controller color].
```

Les touches concernées sont les touches fléchées et la touche [c] pour couleur. Dans cette méthode, bien que les événements clavier soient directement gérés par le *morph*, nous transmettons les interactions utilisateur à l'objet contrôleur. Nous devons définir ces méthodes :

```
BobController>>up
        ^ model move: 0 @ -2
BobController>>down
        ^ model move: 0 @ 2
BobController>>right
        ^ model move: 2 @ 0
BobController>>left
        ^ model move: -2 @ 0
BobController>>color
        ^ model color: Color random
```

En ayant un contrôle de l'application dans un autre objet que la vue, on peut changer le contrôle de l'objet indépendamment de la vue. Par exemple, pour un utilisateur ayant moins de droit, on associera un autre contrôleur bridant les moyens d'interaction.

Le contrôleur de Bob relaie l'information au modèle en précisant toutefois la nature des modifications. Cela nous permet de compléter notre modèle comme suit :

```
BobModel>>move: aPoint
        position := position + aPoint.
        self changed: #move
BobModel>>color: aColor
        color := aColor.
        self changed: #color
```

Dans ces deux dernières méthodes, l'envoi de message `changed:` est une des composantes du pattern Observer. Ce message notifie que `BobModel` a changé, tout en précisant par un symbole le type de changement, sur le **mouvement** ou la **couleur**. Du coté de la vue, qui doit intercepter les notifications, nous implémentons la méthode `udpate:` :

```
BobView>>update: anAspect
        anAspect == #move
                ifTrue: [self updatePosition].
        anAspect == #color
                ifTrue: [self updateColor]
BobView>>updateColor
        self color: model color
BobView>>updatePosition
        self position: model position
```

Mais comment le message `update:` est-il envoyé à la vue ? Eh bien, notre vue doit s'enregistrer en tant qu'objet dépendant du modèle. Nous pouvons le faire lors de l'initialisation de la variable d'instance `model` de la vue :

```
BobView>>model: aModel
        model
                ifNotNil: [model removeDependent:
self].
        model := aModel.
        self updatePosition; updateColor.
        model addDependent: self.
        controller model: model
```

L'enregistrement de la dépendance se fait avec l'envoi de message `model addDependent: self`. La vue est ajoutée à une collection référencée dans une variable d'instance de `model`. Pour enlever une dépendance, nous utilisons le message `model removeDependent: self`. Lorsqu'une vue n'est plus utilisée, il est important qu'elle s'enlève des dépendances du modèle. Nous ajoutons cela dans une méthode `release` :

```
BobView>>release
        super release.
        model
        ifNotNil: [model removeDependent: self]
```

Ainsi, pour mettre en musique le pattern Observer, nous utilisons quatre méthodes :

1. `model addDependant: view` pour enregistrer `view` comme objet dépendant de `model` ;
2. `model changed: #aspect` pour notifier un changement dans `model` de type `#aspect` ;
3. `view udpate: anAspect` qui réceptionne la notification de changement ;
4. `model removeDependent: view` pour supprimer `view` de la liste des objets dépendants de `model`.

Il existe une variante de `update:with:` et `changed: with:` qui permet de passer un argument. Regardez la définition de la méthode `changed:` dans la classe `Object`. Vous verrez qu'il est très facile de comprendre comment fonctionne ce mécanisme. Dans ce pattern, il est tout à fait possible d'enregistrer différentes vues à un même modèle. Cela nous permet d'avoir différentes vues toujours synchronisées sur le même modèle, ceci avec un minimum d'effort. Il existe aussi des versions utilisant la génération d'événements plus spécifiques qui utilisent le SASE pattern, c'est-à-dire qui s'enregistre en précisant le message à exécuter en réponse à un *change*.

Pour mettre en mesure le modèle, la vue et le contrôleur, nous définissons une classe `Bob` qui est notre domaine :

```
Object subclass: #Bob
        instanceVariableNames: 'model view stack'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'LinuxMag92'
```

La variable `stack` nous servira dans la suite pour le pattern Command.

L'Extreme Programming ou XP est une méthode de gestion de projet informatique adaptée aux équipes réduites. Elle pousse à l'extrême des principes simples comme la revue de code permanente, les tests systématiques, le refactoring...

```
Bob>>initialize
        super initialize.
        model := BobModel new.
        view := BobView new model: model;
                domain: self.
        view openInWorld
Bob>>model
        ^ model
Bob>>model: aModel
        model := aModel.
        view model: model;
        domain: self
```

Et terminons par une méthode de classe `newOn:` qui nous servira à définir plusieurs instances de `Bob` partageant le même modèle :

```
Bob class>>newOn: aModel
        | instance |
        instance := self new.
        instance model: aModel
```

Puis, pour finir, des méthodes d'accès au domaine dans la vue et le contrôleur :

```
BobView>>domain
        ^ domain
BobView>>domain: aDomain
        domain := aDomain
BobController>>domain
        ^ view domain
```

Maintenant, amusons-nous un peu avec quelques objets. Nous commençons par créer une instance de Bob, dans un *workspace*. Exécutez :

```
bob1 := Bob new
```

Pour contrôler Bob, cliquez dessus. Son contour devient rouge. Puis, déplacez-le avec les touches fléchées, vous pouvez aussi changer sa couleur avec la touche [c]. Maintenant, nous allons faire une chose plus marrante, qui est au cœur de l'apport des patterns MVC et Observer. Exécutez le code suivant :

```
bob2 := Bob newOn: bob1 model
```

Déplacez à nouveau `bob1` avec les touches fléchées. `bob2` se déplace maintenant d'autant. Vous pouvez créer un `bob3` de la même façon et observer le même phénomène.

Ainsi, nous avons plusieurs vues d'un même modèle qui sont complètement synchronisées. Ici, pour la simplicité de la présentation, nous n'avons qu'un seul type de vue, mais il est parfaitement possible d'avoir des vues de types différents représentant un même modèle. Les modifications faites sur n'importe quelle vue se répercutent sur le modèle, et, par le système de notification, sur l'ensemble des vues du modèle. Notez que le pattern MVC a aussi ses détracteurs qui ont introduit MVP pour *Modele View Presenter*, car ils pensent que le contrôleur dépend de la manière dont les données sont présentées. Ainsi, tout le framework graphique de Dolphin Smalltalk est basé sur MVP, ce qui lui confère une manière élégante et simple de composer les applications.

Pattern Command

Nous allons maintenant nous intéresser à quelque chose d'encore plus amusant : implémenter dans Bob un système de défaire/refaire (les fameux `undo/redo` que nous trouvons dans les applications). Pour ce faire, nous nous basons sur le pattern Command.

L'intention du pattern Command est la suivante. Command représente une demande d'exécution comme un objet, de façon à pouvoir manipuler différentes demandes pour, par exemple, créer des logs, faire des queues de demandes ou les défaire.

Notre implémentation du pattern comprend :

1. Une hiérarchie de classes de sommet, la classe abstraite `BobCommand`. Les classes peuplant cette hiérarchie sont autant de types d'ordre que nous pouvons donner à Bob.
2. Une classe `BobCommandStack` qui est une pile de commandes. C'est une instance de cette classe qui nous permet d'enregistrer et de naviguer dans les défaire/refaire.

Définissons d'abord la classe abstraite d'une commande :

```
Object subclass: #BobCommand
        instanceVariableNames: 'model'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'LinuxMag92'
```

Une commande doit savoir sur quel modèle elle agit :

```
BobCommand>>model: aModel
        model := aModel
```

Ensuite, nous souhaitons exécuter et dés-exécuter une commande :

```
BobCommand>>execute
        ^ self subclassResponsibility
BobCommand>>unexecute
        ^ self subclassResponsibility
```

Pour Bob, nous avons deux types d'ordre : déplacement et changement de couleur. Cela correspond donc à deux classes de commande `BobCommandMove` et `BobCommandColor`. Regardons la commande pour le mouvement :

```
BobCommand subclass: #BobCommandMove
        instanceVariableNames: 'delta'
        classVariableNames: ''
        poolDictionaries: ''
        category: 'LinuxMag92'
```

La variable d'instance `delta` correspond à la variation dans le déplacement de Bob. C'est une référence vers une instance de `Point` (c.-à-d. `-2@0`). Ses méthodes sont logiquement définies comme suit :

```
BobCommandMove>>delta: aPoint
        delta := aPoint
BobCommandMove>>execute
        model move: delta
BobCommandMove>>unexecute
        model move: delta negated
```

La définition de la commande de couleur est fort semblable et elle ne pose pas de problème particulier :

MVC a été mis au point en 1979 par Trygve Reenskaug, qui travaillait alors sur Smalltalk dans les laboratoires de recherche Xerox PARC.

```
BobCommand subclass: #BobCommandColor
          instanceVariableNames: 'color oldColor'
          classVariableNames: ''
          poolDictionaries: ''
          category: 'LinuxMag92'
BobCommandColor>>color: aColor
          color := aColor
BobCommandColor>>oldColor: aColor
          oldColor := aColor
BobCommandColor>>execute
          ^ model color: color
BobCommandColor>>unexecute
          ^ model color: oldColor
```

Remarquez que, pour cette commande, nous avons besoin de deux variables d'instance `color` et `oldColor` pour être en mesure de défaire/refaire correctement la commande. Pour l'autre commande, ce n'était pas nécessaire, car nous pouvions revenir à l'état précédent par la négation du `delta`.

Notez qu'en Squeak nous avons aussi une classe qui représente un message et que nous aurions pu utiliser pour représenter une commande.

Maintenant, la définition de la pile de commandes :

```
ReadWriteStream subclass: #BobCommandStack
          instanceVariableNames: ''
          classVariableNames: ''
          poolDictionaries: ''
          category: 'LinuxMag92'
```

La classe est basée sur un flot de données (*Stream*) en lecture-écriture. Sur cette classe, nous redéfinissions la méthode de classe `new`. Nos commandes sont enregistrées dans des tableaux, que nous parcourons avec une *stream* :

```
BobCommandStack class>>new
          ^ self on: Array new
```

L'utilisation d'un flot de données sur un tableau permet d'avoir un curseur pour parcourir cette collection (c.-à-d. tableau) de commande.

Nous avons ensuite besoin d'obtenir, selon la position courante dans la pile, une commande précédente (`previous`) et de placer une nouvelle commande dans la pile, à l'emplacement courant (`nextPut:`) :

```
BobCommandStack>>previous
          self position = 0
          ifTrue: [^ nil].
          self position: self position - 1.
          ^ self peek
BobCommandStack>>nextPut: command
          super nextPut: command.
          self truncate.
          ^ command
BobCommandStack>>truncate
          | oldReadLimit |
          oldReadLimit := readLimit.
          readLimit := position.
          oldReadLimit > readLimit
                    ifTrue: [readLimit
                              to: oldReadLimit
                              do: [:index |
collection at: index + 1 put: nil]]
```

La commande `truncate` utilisée par `nextPut:` permet de couper la queue du flot de données lorsqu'une nouvelle commande est insérée. En effet, si nous ne sommes pas à la fin de la pile et qu'une nouvelle commande est ajoutée, nous devons effacer la fin du flot, car les commandes suivantes ne sont plus cohérentes par rapport à l'état du modèle.

Maintenant, voyons comment les commandes vont s'insérer dans le contexte de notre application Bob. Jusqu'à présent, nos ordres à Bob passaient par l'objet contrôleur, qui, à son tour, donnait l'ordre de modification au modèle. Il semble donc que nous devons insérer nos commandes entre ces deux objets. Nous proposons que :

1. Le contrôleur donne toujours l'ordre de modification, mais en émettant une commande.
2. La commande n'est pas émise vers le modèle. En effet, la gestion des commandes ne concerne pas le modèle. Nous émettrons les messages de commande à destination du domaine, représenté par une instance de l'objet `Bob`.
3. Ensuite, depuis le domaine, nous créons, enregistrons et exécutons les commandes.

Cela se traduit par les modifications et ajouts suivants :

```
BobController>>up
self domain moveCommandFor: model with: 0 @ -2
BobController>>down
self domain moveCommandFor: model with: 0 @ 2
BobController>>right
self domain moveCommandFor: model with: 2 @ 0
BobController>>left
self domain moveCommandFor: model with: -2 @ 0
BobController>>color
self domain
          colorCommandFor: model
          with: Color random
          with: view color
```

Maintenant, dans le domaine, nous définissons les méthodes pour créer les deux types de commande :

```
Bob>>moveCommandFor: aModel with: aPoint
          | command |
          command := BobCommandMove new model: aModel;
                                                delta: aPoint.
          stack nextPut: command.
          command execute
Bob>>colorCommandFor: aModel with: newColor with: oldColor
          | command |
          command := BobCommandColor new model: aModel;
                                color: newColor;
                                oldColor: oldColor.
          stack nextPut: command.
          command execute
```

Ainsi, la commande est construite, enregistrée dans la pile et exécutée.

Dans l'initialisation de Bob, nous devons ajouter l'initialisation de la pile :

```
Bob>>initialize
        ...
        stack := BobCommandStack new.
```

Il nous reste à ajouter l'activation du défaire (*undo*) et refaire (*do*) au clavier avec les touches [u] et [r]. À la fin du message `handleKeystroke: evt` de la classe `BobView`, ajoutez les lignes suivantes :

```
BobView>>handleKeystroke: evt
...
keyValue = 117
                ifTrue: [controller undo].
        keyValue = 114
                ifTrue: [controller redo]
```

Puis, dans les classes `BobController` et `Bob`, ajoutez les méthodes suivantes :

```
BobController>>undo
        ^ self domain undo
BobController>>redo
        ^ self domain redo
Bob>>undo
        | command |
        command := stack previous.
        command
                ifNotNil: [command unexecute]
Bob>>redo
        | command |
        command := stack next.
        command
                ifNotNil: [command execute]
```

Depuis un workspace, créez une instance de Bob, comme nous l'avons fait précédemment, cliquez dessus et déplacez-le avec les touches fléchées. Essayez ensuite les touches [u] et [r] pour manipuler les instructions défaire/refaire.

Conclusion

Nous avons terminé notre exploration de ces trois patterns. Pour vous familiariser avec ceux-ci, nous vous invitons à expérimenter et modifier l'exemple. Vous pouvez télécharger l'exemple complet depuis l'adresse ci-après. Par exemple, vous pouvez ajouter un nouveau type de vue, qui affiche dans un morph la position et la couleur courante du modèle.

Dans un prochain article, nous aborderons d'autres patterns comme le Composite qui permet d'offrir une interface uniforme pour accéder à des structures récursives, le Visitor qui permet de visiter de telles structures et le Singleton qui permet à une classe de spécifier qu'une seule de ses instances n'est active à un moment donné.

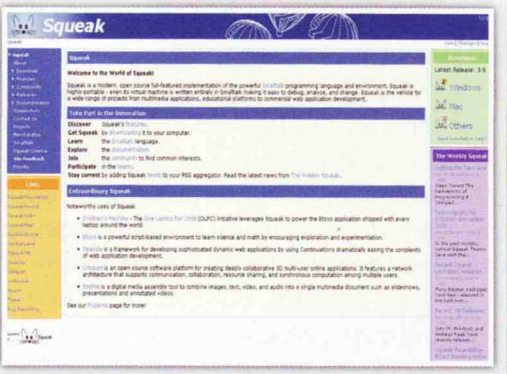
H. Fernandes & S. Ducasse,

hilaire@ofset.org

Stephane.Ducasse@gmail.com

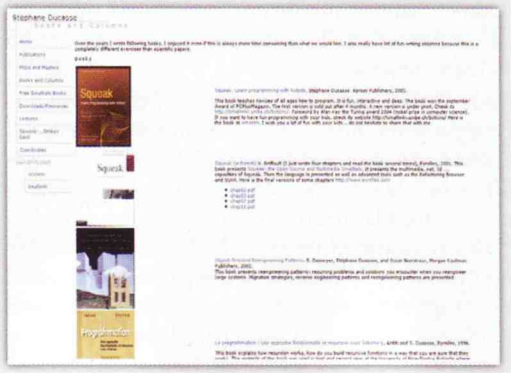
LIENS

- Les codes sources des exemples à télécharger : http://squeak.ofset.org/LM92
- Le site officiel : http://www.squeak.org/



- Le *wiki* de la communauté : http://minnow.cc.gatech.edu/
- Le wiki de la communauté française : http://community.ofset.org/wiki/Squeak
- Le groupe des utilisateurs européens de Smalltalk (*European Smalltalk User Group*). L'adhésion est gratuite : http://www.esug.org/

- Des livres gratuits en ligne sur Smalltalk et Squeak : http://www.iam.unibe.ch/~ducasse/FreeBooks.html



- Un livre sur Squeak en français : BRIFFAULT (X.) et DUCASSE (S.), *Squeak*, Eyrolles, 2002, http://www.iam.unibe.ch/~ducasse/Books.html

Alexandre Courbot

CMake : la relève dans la construction de projets

La compilation (que nous appellerons plus généralement « construction » dans cet article) de projets logiciels un tant soit peu volumineux fait généralement appel à un outil dédié, chargé de vérifier que la compilation est possible, de la paramétrer et enfin de l'effectuer. Le couple autoconf/automake est l'outil le plus connu permettant d'accomplir cette tâche, mais il souffre de certains défauts, dont sa complexité n'est pas le moindre. Nous allons découvrir CMake, qui se veut une alternative plus simple, plus portable et plus élégante.

En effet, si la plupart des projets libres utilisent le célèbre triplet `autoconf/automake/libtool` (ou `autotools`) pour gérer leur compilation (le fameux `./configure; make; make install`), l'utilisation de celui-ci ne s'avère pas des plus aisées... et peut rapidement devenir un véritable casse-tête. CMake prétend remplacer ce vénérable système de construction par un seul outil, plus cohérent, plus performant et également plus simple à mettre en œuvre. Le présent article d'introduction à cet outil s'adresse autant aux personnes maîtrisant les *autotools* qu'à ceux qui n'ont jamais appelé gcc que manuellement. Pour ces derniers, un rappel de la problématique derrière la construction de projets s'impose.

La construction d'un projet, c'est...

La construction d'un projet désigne toutes les actions permettant de transformer le code source d'un projet vers une forme binaire exécutable et adaptée à un système donné. Elle comprend donc bien évidemment la compilation, mais d'autres actions peuvent intervenir : génération de fichiers de configuration, de la documentation, tests de bon fonctionnement, etc.

Sous Unix/Linux, la construction d'un projet est typiquement réalisée par la commande `make`. Celle-ci lit la description du projet depuis un fichier appelé `Makefile`. Un `Makefile` est constitué de cibles, de dépendances nécessaires à la construction de ces cibles, et de leurs règles de construction. À partir de ces éléments, `make` est capable de déterminer, d'une part, les éléments du projet qui nécessitent d'être construits pour obtenir le résultat désiré, et, d'autre part, l'ordre dans lequel ceux-ci doivent l'être. Mieux, une fois relancée, elle ne reconstruit que les parties du projet qui ont été modifiées depuis le dernier appel (ainsi que celles qui en dépendent). `make` est donc un outil essentiel pour peu que votre projet se compose de plus d'un fichier source. Étant disponible sur de nombreuses plateformes (tous les Unix, mais également Windows, MacOS, etc.) cet outil est pratiquement considéré comme un standard pour construire un projet.

Cependant, si `make` est un outil extrêmement répandu, d'autres moyens de construire un projet existent, proposés notamment par certains environnements de développement intégrés. Visual Studio de Microsoft dispose ainsi de son propre constructeur, et KDevelop sa propre gestion de projets basée sur `make`. De plus, un projet repose sur des dépendances (bibliothèques, programmes) qui doivent être présentes pour son bon fonctionnement, voire souvent pour que sa compilation aboutisse. Enfin, le compilateur à appeler et les options de compilation ne sont pas forcément les mêmes selon le système.

Comment déjà gérer les disparités entre plateformes, notamment concernant l'emplacement des dépendances, le compilateur à utiliser et la manière de l'appeler ? Une solution primitive consiste à demander à l'utilisateur de « remplir les trous » dans un `Makefile` existant, ou de commenter/décommenter certaines lignes en fonction de sa plate-forme. Une autre solution est de fournir plusieurs `Makefile` : un par plate-forme supportée. Mais ces solutions n'ont rien de vraiment satisfaisant, car elles ne prennent en compte qu'un nombre limité de cas et demandent une implication trop forte de la part de la personne qui compile le logiciel.

Alors, au-dessus de `make`, s'est greffée une chaîne d'utilitaires permettant de détecter les bons paramètres de compilation et de générer les `Makefile` adaptés en fonction du système hôte et de sa configuration. La plus connue est la fameuse chaîne des `autotools` (`autoconf/automake/libtool`), mais il en existe d'autres (`imake`, etc.). Les autotools sont en fait trois outils écrits pour fonctionner ensemble :

- `libtool` offre une manière portable de créer des bibliothèques ;
- `automake` permet d'écrire des squelettes de `Makefile` portables ;
- `autoconf` permet de décrire les dépendances du projet, de le paramétrer, et génère à partir des squelettes fournis par `automake` des `Makefile` capables d'utiliser `libtool`.

`autoconf` crée en réalité le fameux script *shell* `configure` qui détermine la présence des dépendances et génère les `Makefile` qui seront utilisés pendant la compilation : `configure` crée les `Makefile`, `make` les utilise pour compiler le projet. Ainsi, l'utilisateur n'a pas à se soucier des spécificités de sa plate-forme, qui sont prises en charge par ces outils. Autre avantage pour l'utilisateur, celui-ci n'a pas besoin de disposer des autotools pour compiler un projet les utilisant, étant donné que tout est géré par le script shell `configure`. Tout va donc pour le mieux dans le meilleur des mondes ? Pas tout à fait.

...une galère

Mais alors, quel est le problème avec les autotools ? Tout d'abord, la chaîne des autotools est très loin de former un ensemble cohérent, notamment dans la syntaxe utilisée. Ainsi, `autoconf` convertit le fichier `configure.ac`, écrit dans le langage `m4`, vers le script shell `configure` ; `automake`, écrit en Perl, lit une pseudo-syntaxe de `Makefile` et en produit une autre ; l'utilisation des spécificités de `libtool` nécessite de maîtriser une syntaxe additionnelle. Rien que pour compiler un projet, 4 nouvelles syntaxes à connaître !

Ensuite, `automake` semble suivre une règle d'or : celle de rendre chaque nouvelle version incompatible avec les précédentes. Si bien qu'il n'est pas rare d'avoir 2 ou 3 versions d'`automake` installées sur un même système, pour pouvoir construire différents projets.

Pour terminer, les messages d'erreur de chacun de ces outils sont bien évidemment propres à chacun et cryptiques au possible, si bien que les autotools se sont vus affublés du sobriquet de « *autohell* ». Ces problèmes ont contribué à qualifier la construction de projets de processus pénible et difficile, accessible uniquement à quelques gourous. Mais en y réfléchissant un peu... est-ce si compliqué que ça de construire un projet ?

CMake

CMake fait figure d'outsider aux autotools pour construire de projet. Il dispose d'une communauté de développeurs active, enrichie de nombreux utilisateurs. Récemment, un événement important l'a mis sous le feu des projecteurs : le projet KDE l'a adopté avec succès (et soulagement) pour gérer la construction de la future version 4, en remplacement des autotools. Par rapport aux autotools, CMake se veut :

- Plus simple : là où les autotools demandent de connaître plusieurs syntaxes et d'écrire différents types de fichiers, CMake n'utilise qu'un seul type de fichier. La description des cibles est également plus concise.
- Plus pratique : une seule commande (`cmake`) remplace les invocations successives de `automake`, `autoconf` et du script `configure`.
- Plus rapide : l'exécution de CMake surpasse largement en vitesse celle d'un script `configure`.
- Plus *user-friendly* : l'utilisateur n'a que faire des commandes de compilation de 15 lignes qui remplissent son terminal. Les `Makefile` générés par CMake informent de la progression de la construction par un message sur la ressource actuellement construite, ainsi qu'un pourcentage de complétion.
- Plus portable : tout comme les autotools, CMake sait bien entendu générer des `Makefile`, mais peut aussi produire des fichiers de projet exploitables par KDevelop et Microsoft Visual Studio.

Beaucoup de promesses donc, mais voyons jusqu'à quel point elles sont tenues au travers d'un petit exemple. Nous travaillerons ici avec la version 2.4 de CMake.

Exemple pratique

Avec les autotools, la configuration de la construction d'un projet se fait au travers de plusieurs fichiers. CMake, au contraire, centralise tout dans un seul fichier, nommé `CMakeLists.txt`. Il contient les directives permettant de s'assurer que toutes les dépendances sont présentes, ainsi que les définitions des bibliothèques et programmes à construire. Nous allons voir comment tout cela fonctionne avec un petit exemple de projet, très basique, définissant une bibliothèque fournissant une implémentation de la fonction `ceil` (qui retourne l'arrondi supérieur d'un nombre flottant) et d'un programme l'utilisant pour obtenir l'arrondi supérieur d'un nombre passé en argument.

Code du programme

Notre « projet » s'articulera donc autour des fichiers suivants : tout d'abord, la bibliothèque `myceil`, définie dans un sous-répertoire `lib`, comprenant l'implémentation de `ceil` définie dans `lib/my_ceil.c` :

```
#include <math.h>
double my_ceil(double d) {
  if (d - floor(d) != 0.0) d += 1.0;
  return floor(d);
}
```

Ensuite, la définition de l'interface de notre bibliothèque dans `lib/my_ceil.h` :

```
#ifndef __MY_CEIL_H_
#define __MY_CEIL_H_
extern double my_ceil(double d);
#endif
```

Et enfin, notre programme interactif dans `main.c` :

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "lib/my_ceil.h"

int main(int argc, char * argv[]) {
  double d = atof(argv[1]);
  printf("%f\n", my_ceil(d));
  return EXIT_SUCCESS;
}
```

Une fois compilé et lié avec la bibliothèque `myceil`, ce programme s'invoque en lui passant un nombre flottant en argument.

Tout ceci est bien évidemment très basique (il n'y a ainsi aucun contrôle d'erreur), mais nous étudions l'outil de construction du programme, et non pas le programme en lui-même.

Le fichier CMakeLists.txt

Il nous faut maintenant définir une série de fichiers `CMakeLists.txt` nous permettant de compiler la bibliothèque `myceil`, puis le programme `ceil` et de lier ce dernier avec notre bibliothèque. Un changement dans le code source doit déclencher la recompilation de tous les éléments dont il dépend. Pour corser le jeu, nous allons en plus exiger que le programme soit obligatoirement lié avec la `zlib`, et avec les bibliothèques `X11` si celles-ci sont disponibles. Ces

CMake n'est pas la seule alternative aux Autotools. Le projet Scons est également un candidat intéressant.

dépendances ne sont bien sûr pas indispensables pour notre projet, mais nous permettront d'étudier d'autres fonctionnalités de CMake.

Voici à quoi ressemble notre fichier `CMakeLists.txt` en racine du projet :

```
PROJECT(ceil)
```

En premier lieu, il faut donner un nom à notre projet. Dans notre cas, il s'agira de « ceil ». Notez que la syntaxe des commandes CMake n'est pas sensible à la casse. En revanche, les noms de variables le sont.

La ligne suivante s'assure de la présence de la `zlib` et fait échouer la configuration du projet si celle-ci n'est pas présente grâce au mot-clé `REQUIRED` :

```
FIND_PACKAGE(ZLIB REQUIRED)
```

Même chose pour les bibliothèques X11, mais celles-ci sont optionnelles :

```
FIND_PACKAGE(X11)
```

Comme nous utilisons la `zlib`, le chemin de ses en-têtes doit être fourni au compilateur. La ligne suivante permet d'obtenir cet effet :

```
INCLUDE_DIRECTORIES(${ZLIB_INCLUDE_DIR})
```

Notez comment la variable `ZLIB_INCLUDE_DIR`, définie par `FIND_PACKAGE`, est utilisée.

Concernant les bibliothèques X11, il y a deux possibilités : soit elles sont présentes sur le système, et, dans ce cas, nous voulons également que le compilateur ait accès aux en-têtes, soit elles ne le sont pas et il n'y a rien à faire. Dans les deux cas, nous allons afficher un petit message pour informer l'utilisateur de leur présence ou pas, ce qui nous permettra d'étudier la syntaxe quelque peu inhabituelle du `IF` de CMake :

```
IF(X11_FOUND)
  MESSAGE(STATUS "X11 présent")
  INCLUDE_DIRECTORIES(${X11_INCLUDE_DIR})
ELSE(X11_FOUND)
  MESSAGE(STATUS "X11 absent")
ENDIF(X11_FOUND)
```

Voilà pour nos dépendances. Nous pouvons maintenant définir notre projet. Comme nous l'avons déjà dit, il se compose d'une bibliothèque partagée (qui s'appellera `myceil` et se situe dans le sous-répertoire `lib`) et d'un exécutable `ceil`.

La compilation de la bibliothèque s'effectuant dans un sous-répertoire, il nous faudra définir un autre fichier `CMakeLists.txt` à l'intérieur de ce sous-répertoire, et demander à CMake de le parcourir. Pour cette dernière opération, il nous suffit d'appeler la commande `ADD_SUBDIRECTORY` dans notre `CMakeLists.txt` racine :

```
ADD_SUBDIRECTORY(lib)
```

Elle demande à CMake d'évaluer le répertoire passé en paramètre et de prendre en compte toutes ses directives. Nous allons tout de suite écrire le contenu du fichier `lib/CMakeLists.txt`, qui ne fera que définir les règles de construction de notre bibliothèque.

Commençons par définir une variable contenant les sources de la bibliothèque `myceil`. Pour cela, nous utilisons la commande `SET`. Son premier argument est le nom de la variable, les suivants constituent sa valeur :

```
SET(myceil_lib_src my_ceil.c)
```

Si nous avions voulu spécifier plusieurs fichiers sources, nous les aurions ajoutés après le premier en utilisant un espace comme séparateur.

La définition de la bibliothèque partagée `myceil` se fait ensuite naturellement de la façon suivante, en utilisant la variable que nous venons de déclarer :

```
ADD_LIBRARY(myceil SHARED ${myceil_lib_src})
```

Et voilà, en deux lignes les règles de construction de notre bibliothèque sont définies. Si nous avions voulu produire une bibliothèque statique, nous aurions utilisé le mot clé `STATIC` en lieu et place de `SHARED`. Maintenant, retour à notre `CMakeLists.txt` racine pour définir les règles de construction de notre exécutable `ceil`. Nous pouvons également passer par une variable pour définir ses fichiers sources, et utiliser la commande `ADD_EXECUTABLE` de la même manière que nous avons utilisé `ADD_LIBRARY` :

```
SET(ceil_src main.c)
ADD_EXECUTABLE(ceil ${ceil_src})
```

Une dernière chose : `ceil` doit être lié avec notre bibliothèque `myceil`, la bibliothèque mathématique, la `zlib`, et les bibliothèques X11 si ces dernières sont présentes. La commande `TARGET_LINK_LIBRARIES` permet de spécifier avec quels autres éléments un exécutable ou une bibliothèque doit être lié :

```
TARGET_LINK_LIBRARIES(ceil myceil m ${ZLIB_LIBRARY}
${X11_LIBRARIES})
```

Et c'est tout ! Si les bibliothèques X11 ne sont pas présentes, la variable `X11_LIBRARIES` sera vide, et n'aura donc aucun effet sur l'édition des liens.

Construction du projet

Nous pouvons maintenant essayer tout cela. Dans un répertoire, vous aurez créé les fichiers nécessaires :

```
$ ls
CMakeLists.txt  lib/  main.c
$ ls lib
CMakeLists.txt  my_ceil.c  my_ceil.h
```

Lancez la commande `cmake .` pour générer le `Makefile` correspondant à notre projet :

```
$ cmake .
-- Check for working C compiler: gcc
-- Check for working C compiler: gcc - works
-- Check size of void*
-- Check size of void* - done
-- Check for working CXX compiler: c++
-- Check for working CXX compiler: c++ -- works
-- Found ZLIB: /usr/lib/libz.so
-- Looking for XOpenDisplay in /usr/lib/libX11.so;/usr/lib/libXext.so
-- Looking for XOpenDisplay in /usr/lib/libX11.so;/usr/lib/libXext.so - found
```

```
-- Looking for gethostbyname
-- Looking for gethostbyname - found
-- Looking for connect
-- Looking for connect - found
-- Looking for remove
-- Looking for remove - found
-- Looking for shmat
-- Looking for shmat - found
-- Looking for IceConnectionNumber in ICE
-- Looking for IceConnectionNumber in ICE - found
-- X11 présent
-- Configuring done
-- Generating done
-- Build files have been written to: /home/me/CMake/exemple
```

Vous aurez remarqué que tous les tests ont été effectués très rapidement. Notez également l'affichage de notre petit message indiquant que les bibliothèques X11 ont été trouvées. Si ce n'est pas le cas chez vous, ce n'est pas un problème pour la suite de la construction du projet.

Outre un `Makefile`, l'invocation de CMake a créé d'autres fichiers à son usage. Peu importe, nous pouvons maintenant compiler notre projet avec `make` :

```
$ make
Scanning dependencies of target myceil
[ 50%] Building C object lib/CMakeFiles/myceil.dir/my_ceil.o
Linking C shared library libmyceil.so
[ 50%] Built target myceil
Scanning dependencies of target ceil
[100%] Building C object CmakeFiles/ceil.dir/main.o
Linking C executable ceil
[100%] Built target ceil
```

Vous pouvez maintenant essayer votre programme :

```
$ ./ceil 1.2
2.000000
```

Les habitués aux autotools seront surpris par la concision du rapport de compilation : juste un pourcentage de progression et une description de l'action courante. Est-il besoin de plus ? Parfois oui, pour déboguer les problèmes de compilation. Dans ce cas, il suffit de déclarer la variable d'environnement `VERBOSE` à 1 avant de lancer `make` pour obtenir les commandes de construction exécutées :

```
$ VERBOSE=1 make
```

Les classiques `make clean` et `make install` sont également de la partie. Pour avoir une liste des cibles définies dans votre `Makefile`, vous pouvez invoquer `make help`.

Et comme nous l'avons dit, CMake sait générer autre chose que des `Makefile`. Vous êtes fan de Kdevelop ? Invoquez CMake avec l'option `-G` et les fichiers de projet seront générés pour vous !

Paramétrage de la compilation

Nous avons fait une invocation simple et rapide de CMake pour générer les `Makefile` de notre projet. Cette invocation convient pour compiler notre programme et le lancer, mais ce cas d'utilisation est loin d'être unique.

La sortie générée par CMake peut être paramétrée par de nombreuses variables d'environnement. Celles-ci sont définies selon le même modèle que les définitions de GCC, au travers de l'option `'-DNOM_VARIABLE=VALEUR'`. Parmi les options les plus fréquentes, on peut lister :

- `CMAKE_BUILD_TYPE` peut prendre les valeurs `Debug` (produit un binaire contenant les symboles de debug) ou `Release` (produit un binaire optimisé et sans symboles de debug).
- `CMAKE_INSTALL_PREFIX` pointe vers le chemin d'installation du projet (par défaut `/usr/local`).

Vous avez également la possibilité de définir des paramètres de compilation propres à votre projet. Dans le fichier `CMakeLists.txt`, la commande `OPTION` permet de définir des options de type booléen :

```
OPTION(WITH_GUI "Compiler l'interface graphique" OFF)
```

Le premier paramètre est le nom de la variable définissant l'option, le second la description de cette option, et le troisième la valeur par défaut. Plus loin dans le fichier, il est possible de voir si l'option a été activée en testant la valeur de la variable déclarée :

```
IF(WITH_GUI)
  MESSAGE(STATUS "Compilation de l'interface graphique activée")
ENDIF(WITH_GUI)
```

D'une manière plus générale, vous pouvez vérifier la présence et la valeur de n'importe quelle variable arbitraire qui aurait été passée à CMake. Par exemple, si nous voulons pouvoir donner le chemin d'un répertoire de données à notre projet tout en assurant une valeur par défaut, nous pouvons utiliser les lignes suivantes :

```
IF (NOT DATA_DIR)
  SET(DATA_DIR "/usr/share/mydatadir")
ENDIF(NOT DATA_DIR)
MESSAGE(STATUS "Données situées dans ${DATA_DIR}")
```

Enrichi de ce code, notre projet peut maintenant se paramétrer avec les deux variables nouvellement déclarées :

```
$ cmake -DWITH_GUI=ON -DDATA_DIR=/home/me/datadir .
```

Passage des paramètres de compilation au projet

Il peut s'avérer nécessaire de conditionner la compilation de certaines parties d'un fichier source en fonction des paramètres de compilation. Par exemple, dans un des sources de notre projet, nous pourrions vouloir ne compiler du code que si l'interface graphique doit être compilée, ou si les bibliothèques X11 sont présentes. Il nous faudrait un fichier d'en-tête qui reflète la partie utile du paramétrage de la construction.

C'est à cela que sert la commande `CONFIGURE_FILE`. Elle prend en paramètres un fichier de modèle, ainsi que la destination du traitement de ce fichier. Voyons comment elle fonctionne par l'exemple. Dans notre

projet, nous définissons un fichier config.h.cmake dont le contenu est le suivant :

```
#ifndef CONFIG_H__
#define CONFIG_H__

#cmakedefine WITH_GUI 1
#cmakedefine DATA_DIR "${DATA_DIR}"
#cmakedefine X11_FOUND 1

#endif
```

Il suffit ensuite de rajouter la ligne suivante dans le fichier CMakeLists.txt :

```
CONFIGURE_FILE(${CMAKE_SOURCE_DIR}/config.h.cmake
${CMAKE_BINARY_DIR}/config.h)
INCLUDE_DIRECTORIES(${CMAKE_BINARY_DIR})
```

Et lors de l'invocation à CMake, le fichier config.h contenant nos déclarations utiles est créé et peut être inclus dans n'importe lequel de nos sources :

```
#ifndef CONFIG_H__
#define CONFIG_H__

#define WITH_GUI 1
#define DATA_DIR "/usr/share/mydatadir"
#define X11_FOUND 1

#endif
```

Pourquoi une nouvelle invocation à INCLUDE_DIRECTORIES après l'appel à CONFIGURE_FILE ? Le fichier config.h.cmake fait partie de notre projet et est logiquement situé dans son répertoire source. Par contre, le fichier config.h généré est spécifique à notre compilation et doit être placé dans le répertoire de construction... qui n'est pas forcément le même que le répertoire des sources et doit donc être indiqué au compilateur si celui-ci veut y trouver des fichiers d'en-tête. Il est en effet possible de séparer répertoire source et répertoire de construction.

Construction séparée

CMake permet sans problème de compiler un projet en dehors de son répertoire source. Pour compiler notre programme précédent, vous auriez pu vous placer dans un répertoire vide et invoquer cmake de la façon suivante :

```
$ cmake /chemin/vers/mon/projet
```

Il suffit ensuite d'appeler make pour compiler le projet sans que le répertoire des sources ne soit affecté. Ceci est particulièrement utile si vous voulez produire différentes versions binaires d'un même projet (une version « debug » et une version « release », par exemple). Il s'agit en fait d'une bonne pratique générale.

Installation du projet construit

Tel que nous avons décrit notre projet, la commande make install n'installera strictement rien. Il nous faut définir dans les fichiers CMakeLists.txt ce que nous désirons installer, et où. La commande INSTALL sert précisément à cela.

```
INSTALL(TYPE fichiers a installer DESTINATION destination)
```

TYPE renseigne sur le type de fichier à installer. Parmi les valeurs possibles, TARGETS indique que nous souhaitons installer des cibles construites par CMake, et FILES sert pour installer des fichiers réguliers. Vient ensuite la liste des cibles ou fichiers à installer, le mot clé DESTINATION, et finalement la destination proprement dite. Si cette dernière désigne un chemin relatif, celui-ci le sera par rapport au préfixe d'installation (défini par la variable CMAKE_INSTALL_PREFIX).

Voici comment nous pouvons compléter notre projet ceil avec ses cibles d'installation. Dans CMakeLists.txt, nous ajoutons une ligne pour installer le programme ceil dans le répertoire bin du préfixe d'installation :

```
INSTALL(TARGETS ceil DESTINATION "bin")
```

Et dans lib/CMakeLists.txt, nous ajoutons deux cibles : une pour installer le fichier d'en-tête (qui n'est pas généré par CMake) dans include, et une autre pour installer la bibliothèque construite par CMake dans lib.

```
INSTALL(FILES my_ceil.h DESTINATION "include")
INSTALL(TARGETS myceil DESTINATION "lib")
```

La commande make install donne maintenant le résultat attendu !

Conclusion

Cet article ne se veut en rien exhaustif : il avait pour but de démontrer l'obsolescence des autotools qui, après avoir bravement rempli leur mission pendant de nombreuses années, ont bien mérité une mise à la retraite. Nous n'avons effectué qu'une introduction rapide à CMake afin de montrer sa simplicité et sa logique, mais CMake est en réalité un outil très puissant et personnalisable qui peut également gérer les tests unitaires (avec ctest) et le *packaging* des projets (avec cpack). Vous pouvez trouver plus d'informations à ces sujets sur son site officiel.

Gageons et espérons que l'adoption de ce nouvel outil de construction par des projets aussi connus que KDE contribue à son utilisation quotidienne, ce qui aura sans nul doute comme effet de rendre la construction de projets plus simple, plus sûre, plus portable et surtout... moins effrayante.

LIENS

- Site officiel de CMake : http://www.cmake.org
- Le code complet de l'exemple de l'article : http://www.gnurou.org/documents/linuxmag/cmake/exemple.tar.gz

Alexandre Courbot,
gnurou@gmail.com

misc

Multi-System & Internet Security Cookbook

En KIOSQUE
Le 2 mars

NUMÉRO 30

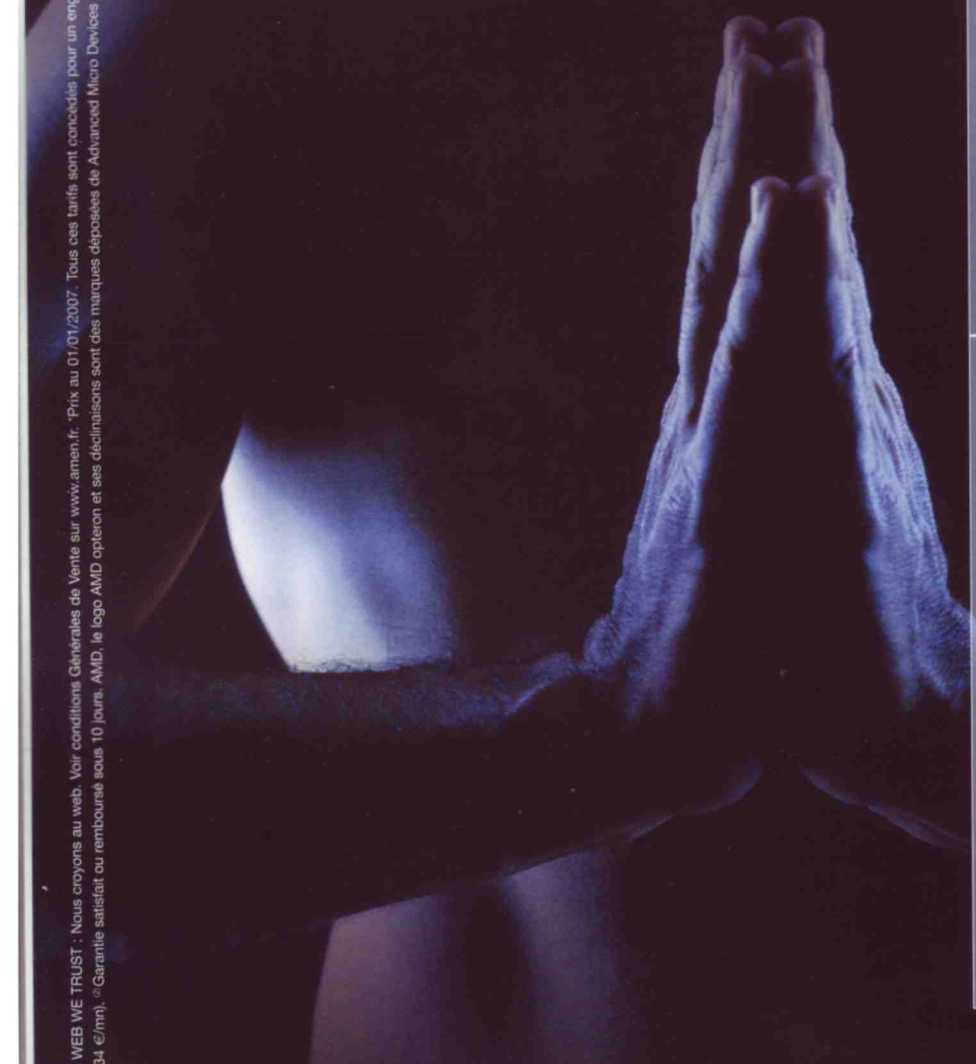
sous réserve de modification

NOMS DE DOMAINE - HÉBERGEMENT - CRÉATION DE SITE - E-COMMERCE - RÉFÉRENCEMENT - BACKUP

amen
IN WEB WE TRUST

serveurs dédiés DUO

Vous n'avez pas à nous prier pour vous offrir deux fois plus de performance !



NOUVEAU
Serveurs dédiés DUO

AMD 64 Opteron

Pour les professionnels les plus exigeants, AMEN lance la nouvelle gamme de serveurs dédiés DUO basée sur des processeurs double coeur, disques durs en RAID, pour vous offrir 2 fois plus de puissance.

DUO 1000 ► **99 € ht/mois***
(118,40 € ttc/mois*)

AMD Opteron 1210 - 2x1,8GHz - RAM 1GB
Disque dur 2x160GB - Raid Soft
2 adresses IP - Interface Plesk 8 jusqu'à 100 domaines - Trafic illimité

DUO 2000 ► **149 € ht/mois***
(178,20 € ttc/mois*)

AMD Opteron 1212 - 2x2,0GHz - RAM 2GB
Disque dur 2x200GB - Raid 1 matériel
4 adresses IP - Interface Plesk 8 jusqu'à 300 domaines - Trafic illimité

DUO 4000 ► **199 € ht/mois***
(238,00 € ttc/mois*)

AMD Opteron 1214 - 2x2,2GHz - RAM 4GB
Disque dur 2x250GB - Raid 1 matériel
6 adresses IP - Interface Plesk 8 jusqu'à 300 domaines - Trafic illimité

Compatibles &

Nous avons foi en un idéal de services, surtout lorsqu'il vous permet de bénéficier des dernières avancées techniques : architecture réseau redondée, bande passante dédiée 2GB, haute disponibilité (99,9%), assistance technique par mail et téléphone 6j/7[1]. Quant à notre 'Garantie satisfait ou remboursé'[2], elle vous permettra d'atteindre la sérénité absolue. **Si vous croyez au web, vous croirez en nous.**

► Pour plus de renseignements **0 892 55 66 77** (0,34€ / min) ou **www.amen.fr**

AMEN RCS PARIS : B 421 527 797. IN WEB WE TRUST : Nous croyons au web. Voir conditions Générales de Vente sur www.amen.fr. *Prix au 01/01/2007. Tous ces tarifs sont concédés pour un engagement annuel. [1]Du lundi au samedi de 9h à 18h au 0899 70 9001 (1,34 € l'appel puis 0,34 €/mn). [2]Garantie satisfait ou remboursé sous 10 jours. AMD, le logo AMD opteron et ses déclinaisons sont des marques déposées de Advanced Micro Devices Inc.