N°165 NOVEMBRE 2013

GNU LINUX MAGAZINE / FRANCE

CLOUD / REPÈRE

Derrière le jargon sans signification informatique réelle, apprenez et comprenez ce qu'est vraiment le cloud, SaaS, PaaS, IaaS… p.46

ADMINISTRATION ET DÉVELOPPEMENT SUR SYSTÈMES OPEN SOURCE ET EMBARQUÉS

NEWS / TOOLKIT

Découvrez les nouveautés de Tcl/Tk 8.6, programmation orientée objet, stackless et bien plus… p.04

NUAGE / LIBRE

OpenStack, une solution open source « clé en main » de plateforme et d'infrastructure de services Cloud p.52

ANDROID / PLUGINS

Ajouter des extensions de layout à une application Android pour modulariser votre création p.62

OS / MOBILE

Créez facilement une application pour Tizen, la plate-forme mobile GNU/Linux HTML 5 à venir p.12

L 19275 - 165 - F: 7,90 € - RD

BARBUS / IOCCC

Étude en détail de la calculatrice multifonction de Hou Qiming : mais où est le code ? p.36

OBJET / CODE

Facilitez vos développements avec la mise en place d'un système de tags sur les objets ELF p.80

RÉSEAU / ANNUAIRE

Des services à la pelle, des clients partout et des informations éparpillées sur les serveurs… Il est temps de faire du rangement !

INSTALLEZ VOTRE SERVEUR LDAP

p.20

1. Centraliser les informations avec LDAP
2. Installer et configurer son serveur slapd
3. Renforcer la sécurité via TLS/SSL/SASL
4. LDAPifier les applications clientes et les configurations

CPP / STL

Repartons sur de bonnes bases avec la bibliothèque standard C++ en (re)découvrant la STL d'aujourd'hui p.70

HUMEUR / CODE

Tout le monde utilise les patrons de conception / design patterns, et vous aussi, la preuve ! p.44

France METRO : 7,90 € – CH : 13 CHF – BEL/PORT.CONT : 8,90 € – DOM TOM : 8,50 € – CAN : 14 $ cad – TUNISIE : 18 TND – MAR : 100 MAD

EN ROUTE POUR LES NOUVEAUX

DOMAINES

Vous pouvez maintenant choisir parmi **plus de 700 nouvelles extensions de domaines** pour créer l'adresse originale et facile à retenir qui vous correspond le mieux, comme par exemple **monrestaurant.paris**, **magalerie.art** ou **legrand.shop**. Pré-réservez aussi de nouvelles extensions pour vos domaines existants. Il sera d'autant plus facile de vous trouver !

Avec environ 20 millions de domaines enregistrés, 1&1 est leader du marché de l'enregistrement de noms de domaine en Europe. Grâce à la **fonction de redirection intégrée**, les domaines enregistrés auprès de 1&1 peuvent rapidement et facilement être redirigés vers n'importe quel site Web, quel que soit votre hébergeur.

Plus d'informations sur **1and1.fr**

NOUVEAU !
PRÉ-RÉSERVEZ SANS FRAIS
ET SANS ENGAGEMENT*

1&1

MEMBRE DE united internet

DOMAINES | MAIL | HÉBERGEMENT | E-COMMERCE | SERVEURS

1and1.fr

* La phase de pré-réservation est gratuite et sans engagement. La disponibilité et le prix des domaines dépendent du registre compétent : 1&1 s'engage à communiquer ces informations dès qu'elles seront disponibles. L'enregistrement du domaine est soumis aux conditions générales de vente de 1&1 et à celles du registre compétent. 1&1 ne peut garantir l'attribution finale des domaines pré-réservés.

SOMMAIRE N°165

NEWS

04 Les nouveautés de Tcl/Tk 8.6

12 Créer une application tizen de app a zen

EN COUVERTURE

20 **On a perdu le Minitel mais on a toujours l'annuaire... LDAP**

L'accès centralisé aux informations est un graal que de nombreuses générations d'administrateurs système ont rêvé d'atteindre. Diverses solutions sont apparues sur le marché mais la plus répandue actuellement s'intitule Lightweight Directory Access Protocol ou encore LDAP pour les intimes. Nous allons essayer de lever un coin du voile sur ce mystérieux et ô combien effrayant service.

REPÈRES

36 Le coin du vieux barbu : Une calculatrice pleine de surprises !

44 La conception de logiciels à l'aide de design patterns

46 La tête dans les nuages ...

NETADMIN

52 OpenStack : Libre comme un Nuage

MOBILITÉ

62 Android : Plugin de layout

CODE(S)

70 C++ Standard Library / STL repartons sur de bonnes bases

80 Mise en place d'un système de tag sur les objets ELF

ABONNEMENTS

17/18/43 Bons d'abonnement et de commande

+ ENCART JETÉ 1&1

GNU/Linux Magazine France
est édité par Les Éditions Diamond

LES ÉDITIONS DIAMOND

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

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

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

Périodicité : Mensuel
Prix de vente : 7,90 €

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

Membre April www.april.org

ÉDITORIAL

Oh mon dieu ! Usenet est mort !

Oui... non, pas vraiment, mais en fait si... Tout dépend de quel point de vue on se place et de la manière dont on défini « mort ». Un simple coup d'œil aux statistiques affichées par la page anglaise de Wikipédia montre une progression du trafic quotidien entre 1996 et nos jours. En termes de volumes on est passé de 4 Go en à quelque chose comme 11 To en 2013... par jour ! Le nombre de posts journaliés à suivi la même progression...

Pourtant, en ressortant ma vieille configuration Slrn et en la dépoussiérant, l'espace d'un moment, j'ai eu un gros doute sur la qualité du serveur NNTP auquel j'accédai. Étrangement les quelques 400 posts sur fr.comp.lang.c dataient pour la plupart de plus d'un an et étaient presque majoritairement constitués du classique post de la charte du groupe... Après vérification de la configuration, espérant vraiment avoir fait une erreur, non, ce bon vieux client Usenet m'affichait bel et bien la liste des posts existants. Un script, lui, devait toujours être à son poste tout comme quelques furieux (sans doute de mon espèce). Avec un regain d'espoir je me jetai sur *fmbl*, prêt à répondre à une question technique et enchaîner sur une recette de cuisine dans la foulé (punition standard), juste pour me prouver que la vie était encore là... mais...

Êtes-vous déjà aller dans une zone industrielle un dimanche histoire de profiter de tout cet espace pour votre petit bolide radiocommandé par exemple ? Vous connaissez alors cette étrange sensation de vide, de désertification et d'abandon. Un grand espace... résonnant encore d'une activité intense, récente, comme l'écho d'un son depuis éteint... Usenet ressemble à cela et entre vous et moi, ça fiche un coup, un sale coup.

La masse de posts et le trafic ? Ce n'est pas une surprise, c'est le flux des données segmentées pour l'échange des médias rippés et encodés (musique, DVD, Bluray, etc). Ce qui est triste, ce n'est pas cette fausse joie, c'est la transformation de Usenet en support d'échange de données binaires, en réseau M2M, alors qu'il s'agissait initialement du plus important réseau d'échange hiérarchisé d'idées. Ceci est arrivé au point où l'accès à Usenet est littéralement devenu synonyme de « copie illégale » et que les providers n'hésitent plus à cesser simplement de proposer ce service. Bien sûr ! Pour le néo-geek dégoulinant de médiocrité et d'inculture ne sachant pas s'exprimer autrement qu'en langage SMS, les newsgroups (Usenet, il ne connait pas) ne sont qu'un dépotoir à médias pirates, rien de plus. Pourquoi simplement se limiter à couper l'accès aux groupes « binaires », personne d'autre n'utilise Usenet, après tout. Non ?

Le pire c'est que ce n'est même plus tout à fait faux. Usenet n'est pas réellement mort, mais il n'est plus vraiment vivant non plus. Et je suis tout aussi responsable que n'importe quel ancien utilisateur Usenet, ce qui ne fait qu'augmenter ma tristesse. Je me suis détourné un temps du réseaux et maintenant qu'il me manque, je retrouve un mort-vivant, un zombie aussi loquace et actif qu'un mollusque dans une cabine à UV...

Il y a plus triste que de voir disparaître un tel lieu de discussion, c'est de le voir transformé en un simple support de transfère de données pour peignes-zizi s'autogratifiant du titre de geek ou de hacker parce qu'il savent pondre un NZB tout en ignorant fièrement les différences entre UUencode, Yenc et Base64. Ça fait mal, vraiment mal.

L'*Eternal September* est un lointain passé désormais et si nous n'y prenons pas garde, d'autres choses suivront le même chemin, soyez-en assuré...

Denis Bodor

LES NOUVEAUTÉS DE TCL/TK 8.6

par Gérard Sookahet

La version 8.6 de Tcl/Tk est sortie fin décembre 2012. Même après deux décennies d'existence, ce langage continue d'évoluer ... à son rythme. Si'l y a deux expressions à retenir parmi les nouveautés les plus visibles, ce sont : programmation orientée objet et stackless. Mais ce n'est pas tout !

1 Introduction

Inventé par John Ousterhout en 1988, Tcl/Tk est un langage de script (Tcl) associé à un toolkit IHM (Tk). Pour comprendre la syntaxe, il suffit de maîtriser 12 règles **[DODECA]**. Doté d'un compilateur de bytecode et d'un mécanisme de runtime avec un système de fichier virtuel dénommé Starkit **[STARKIT]**, il a été Influencé à l'origine par les langages C, Lisp, Shell, Awk et par Hypercard pour Tk. Cependant il intègre au fur et à mesure les évolutions d'autres langages plus récents. La version 8.6 est le résultat de l'implémentation de 50 TIPs (Tcl Improvement Proposals - documents de projet d'amélioration). Hormis le support natif de la programmation orientée objet dans le noyau de Tcl, on note dans le catalogue des nouveautés une couche d'abstraction de base de données (Tcl Database Connectivity), la gestion des exceptions, le support de zlib dans le noyau, les coroutines et la refonte du mécanisme interne d'exécution (Non-Recursive-Engine) au niveau de la pile du langage C.

2 Installation de Tcl/Tk 8.6

Télécharger les fichiers tcl8.6.0-src.tar.gz et tk8.6.0-src.tar.gz sur : http://www.tcl.tk/software/tcltk/download.html.

Et décompresser les deux archives :

```
tar -zxvf tcl8.6.0-src.tar.gz
tar -zxvf tk8.6.0-src.tar.gz
```

Ce qui va créer deux répertoires **/tcl8.6.0** et **/tk8.6.0**.

Ensuite passons à la compilation en tenant compte des versions 64 bits :

```
cd ./tcl8.6.0/unix
./configure --prefix=/usr --mandir=/usr/share/man \
$([ $(uname -m) = x86_64 ] && echo --enable-64bit)
make
```

Si vous envisagez plus tard de compiler des extensions qui dépendent des sources, un passage par **sed** vous sera bien utile :

```
sed -e "s@^\(TCL_SRC_DIR='\).*@\1/usr/include'@" \
    -e "/TCL_B/s@='\(-L\)\?.*unix@='\1/usr/lib@" \
    -i tclConfig.sh
```

Et pour bien faire les choses en tant que **root** :

```
make install
make install-private-headers
ln -v -sf tclsh8.6 /usr/bin/tclsh
chmod -v 755 /usr/lib/libtcl8.6.so
```

Faisons de même pour Tk :

```
cd ../..
cd ./tk8.6.0/unix
./configure --prefix=/usr --mandir=/usr/share/man \
$([ $(uname -m) = x86_64 ] && echo --enable-64bit)
make
```

Ainsi que :

```
sed -e "s@^\(TCL_SRC_DIR='\).*@\1/usr/include'@" \
    -e "/TCL_B/s@='\(-L\)\?.*unix@='\1/usr/lib@" \
    -i tclConfig.sh
```

Pour parfaire l'installation, en tant que **root** :

```
make install
make install-private-headers
ln -v -sf wish8.6 /usr/bin/wish
chmod -v 755 /usr/lib/libtk8.6.so
```

Finalement faisons un test pour vérifier que la nouvelle version de Tcl /Tk est bien présente.

On lance l'interprète de commande en ligne **tclsh** :

```
% tclsh
% info patchlevel
8.6.0
```

On lance l'interprète de commande en ligne **wish** :

```
% wish
% pack [button .b -text [info patchlevel] -command exit]
```

3 Les nouveautés de Tcl 8.6

3.1 La programmation orientée objet

Le support natif de la programmation orientée objet (POO) avec Tcl/Tk est une très longue histoire. Les premières demandes remontent à 1993 ! Depuis, comme aucun consensus n'avait été trouvé, il a fleuri une dizaine d'extensions OO plus ou moins influencées par CLOS, Smalltalk ou C++ sans compter les implémentations personnelles qui ne servent que dans un seul code. Aujourd'hui avec Tcl/Tk 8.6, la POO est officialisée, standardisée et intégrée au cœur du langage. Il s'agit d'un modèle objet compact reposant sur les classes dont les caractéristiques principales sont :

- l'héritage simple et multiple
- les mixins,
- l'introspection d'objets et de classes.

La syntaxe choisie est la suivante :

```
Objet Méthode argument1 argument2 ....
```

C'est celle qui est déjà utilisée avec le toolkit Tk pour décrire les interfaces graphiques.

Voici un exemple simple :

```
oo::class create DistribuerCartes {

  variable compteur

  constructor {} {set compteur 0}

  method donneCartes {{carte 1}} {
        incr compteur
        for {set i 0} {$i < $carte} {incr i} {
           puts "Voici la carte n° $i pour le joueur $compteur"
        }
  }
}

set joueur [DistribuerCartes new]
$joueur donneCartes 3
```

Résultat :

```
Voici la carte n° 0 pour le joueur 1
Voici la carte n° 1 pour le joueur 1
Voici la carte n° 2 pour le joueur 1
```

Un exemple formel d'utilisation des mixins :

```
oo::class create Voiture {
  method demarrer {} { .... }
  method accelerer {} { .... }
  method freiner {} { .... }
}
set voiture_lambda [Voiture new]
$voiture_lambda demarrer


oo::class create ObjetVolant {
    method decollage {} { .... }
    method prendreAltitude {} { .... }
    method aterrissage {} { .... }
}

set voiture_fantomas [Voiture new]
oo::objdefine $voiture_fantomas mixin ObjetVolant
$voiture_fantomas demarrer
$voiture_fantomas decollage
```

Mots-clefs associés : **oo::class**, **oo::define**, **oo::objdefine**, **oo::object**, **oo::copy**, **my**, **self**, **next**.

Les classes, instances et méthodes sont aussi manipulables au niveau de l'API du langage C.

Dans la version antérieure de Tcl 8.5.x, la POO est disponible en tant que package (**package require TclOO**).

Et pour terminer, la POO est un plus mais pas une obligation. Si vous souhaitez persister dans la programmation procédurale, libre à vous.

3.2 Tcl Database Connectivity

TDBC (Tcl DataBase Connectivity) est une interface unifiée pour accéder aux bases de données SQL comprenant des pilotes pour SQLite, MySQL, PostgreSQL et ODBC.

Connection à une base SQLite3 :

```
package require tdbc::sqlite3
tdbc::sqlite3::connection create db "/path/to/mydatabase.sqlite3"
```

Connection à une base MySQL :

```
package require tdbc::mysql tdbc::mysql::connection create db -user
hal -passwd hal9000 \
-host 127.0.0.1 -database odyssey
```

Interrogation de la base :

```
set mot "Clark"
db foreach res { SELECT prenom FROM auteurs WHERE nom = :mot } {
  puts "Résultat \"[dict get $res prenom]\" dans la table des auteurs"
}
```

A noter que Tcl est distribué avec l'extension SQLite3 prête à l'emploi.

Mots-clefs associés : **tdbc::connection** , **tdbc::mapSqlState**, **tdbc::result-set**, **tdbc::statement**, **tdbc::tokenize**, **tdbc::mysql**, **tdbc::odbc**, **tdbc::postgres**, **tdbc::sqlite3**.

3.3 Non-Recursive-Evaluation-Engine et coroutine

Quand une procédure s'appelle elle-même, la pile du langage C ressemble à un sandwich :

```
TEBC : TclExecuteByteCode()  -> le compilateur de bytecode
TEOV : Tcl_EvalObjv()   -> l'évaluateur de commande
TEBC : TclExecuteByteCode()  -> le compilateur de bytecode
```

La récursion est donc très consommatrice pour la pile du langage C. Dans la version 8.6, le mécanisme interne d'exécution a été complètement repensé afin de minimiser l'utilisation de la pile du langage C. Le fruit du travail de Miguel Sofer ouvre de nouveaux horizons à Tcl/Tk tels qu'une grande profondeur de récursion (limitée par la RAM), la récursion terminale, une meilleure gestion des exceptions, la continuation et un pas vers la programmation concurrente.

Voici une illustration de la récursion terminale (**tailcall**) avec la très emblématique procédure factorielle :

```
proc fact {n {k 1}} {
 if {$n < 2} {return $k}
 tailcall fact [expr {$n - 1}] [expr {$k * $n}]
}
```

Les maniaques de la monoligne peuvent toujours continuer à écrire ceci :

```
proc fact n {expr {$n < 2 ? 1: $n * [fact [incr n -1]]}}
```

Le seuil de récursion (limité à 1000 par défaut) est contrôlable avec la commande **interp recursionlimit**.

L'implémentation des coroutines est gérée par 4 nouvelles commandes : **coroutine**, **yield**, **yieldto** et **info coroutine**.

```
proc decompte n {
 while {$n > 0} {
      yield "Compteur: $n"
      incr n -1
 }
 return "MISE A FEU !"
}
```

```
% coroutine fusee decompte 5
Compteur: 5
% fusee
Compteur: 4
% fusee
Compteur: 3
% fusee
Compteur: 2
% fusee
Compteur: 1
% fusee MISE A FEU !
```

Concernant la commande **yieldto**, elle peut être utilisée pour transférer le contrôle d'une **coroutine** à une autre.

3.4 La gestion des exceptions

Tcl disposait déjà d'une commande pour gérer les erreurs avec **catch**. Pour plus de clarté **try/trap/finally** vient la compléter. Il s'agit d'une syntaxe plus familière que l'on retrouve dans d'autres langages.

```
proc division {x y} {
 try {
    puts "Résultat de la division : $x/$y=[expr {$x/$y}]"
 } trap {ARITH DIVZERO} msg {
    puts "Division par zéro impossible : $msg"
 } trap {ARITH DOMAIN} msg {
    puts "En dehors du domaine de validité : $msg"
} on error msg {
    puts "Autre erreur : $msg"
 }
}
# Test
foreach {x y} {16 4  16 0  16.0 0.0  0 0  0.0 0.0  0 coin} {division $x $y}
```

qui donne ceci :

```
Résultat de la division : 16/4=4
Division par zéro impossible : 16/0 -> divide by zero
Résultat de la division : 16.0/0.0=Inf
Division par zéro impossible : 0/0 -> divide by zero
En dehors du domaine de validité : 0.0/0.0 -> domain error: argument
not in valid range
Autre erreur : 0/coin -> can't use non-numeric string as operand of "/"
```

Un autre exemple avec la gestion de l'ouverture d'un fichier :

```
try {
  set f [open /usr/bin/tempo]
} trap {POSIX EISDIR} {} {
  puts "Impossible d'ouvrir /usr/bin/tempo: c'est un répertoire"
} trap {POSIX ENOENT} {} {
  puts "Impossible d'ouvrir /usr/bin/tempo: il n'existe pas"
} finally {
  close $f
}
```

Le contenu de la clause **finally** sera exécuté dans tous les cas.

3.5 La compression et l'encodage

3.5.1 La compression zlib dans le noyau

La commande **zlib** permet la compression/décompression de données et de flux ainsi que le check-summing. Elle est basée sur la bibliothèque Zlib **[ZLIB]** de Jean-Loup Gailly et Mark Adler.

La compression d'un fichier au format gzip :

```
set file test.txt
set fi [open $file rb]
set header [dict create filename $file time [file mtime $file]
comment "Test gzip"]
set fo [zlib push gzip [open $file.gz wb] -header $header]
fcopy $fi $fo
close $fi
close $fo
```

Le calcul d'un CRC32 :

```
% set a [format %x [expr {[zlib crc32 GNULinuxMag 42] & 0xffffffff}]]
% 5258be05
```

3.5.2 Encodage et décodage de séquences binaires

Les sous-commandes **binary encode/decode** permettent l'encodage et le décodage des séquences binaires. Les formats supportés sont **base64**, **hex** et **uuencode**.

Encodage en paire hexadécimale d'un chaîne de caractère :

```
% binary encode hex "GNU Linux Magazine"
% 474e55204c696e7578204d6167617a696e65
```

Encodage d'un fichier binaire en base 64 avec un formatage 64 caractères par ligne :

```
set f (open fichier.bin rb]
set data [read $f]
close $f
puts [binary encode base64 -maxlen 64 $data]
```

3.6 Les listes et chaînes de caractères

3.6.1 Les listes

Dans le tri des listes avec la commande **lsort**, l'option **-stride** permet de trier des éléments groupés.

```
lsort -stride 2 {slackware 10 ubuntu 24 redhat 32 debian 16 suse 28}
```

Et renverra :

```
debian 16 redhat 32 suse 28 slackware 10 ubuntu 24
```

En effet, le tri est effectué par groupe de deux en prenant comme référence par défaut le 1er élément de chaque groupe. Pour faire un tri selon le second élément il suffit de le mentionner avec l'option **-index** (qui commence à 0) :

```
lsort -stride 2 -index 1 {slackware 10 ubuntu 24 redhat 32 debian 16 suse 28}
```

Qui renvoit cette fois-ci :

```
slackware 10 debian 16 ubuntu 24 suse 28 redhat 32
```

Pour la recherche dans une liste déjà triée avec la commande **lsearch**, l'option **-bisect** permet de trouver le point d'insertion d'un nouvel élément. Cette situation se présente lors d'une interpolation ou une approximation dans une table de données.

Dans l'exemple ci-dessous, le point d'insertion de 6 se trouve après le 3ième élément (la numérotation des listes commence à zéro) :

```
% lsearch -sorted -bisect {1 3 5 7 9 11} 6
% 2
```

La commande **lmap** permet de parcourir tous les éléments d'une liste ou plusieurs listes en y effectuant des modifications. Elle remplace avantageusement une boucle **foreach** sur les éléments de la liste.

```
set l [list 1 2 3 4 5 6 7 8 9]
set lc [lmap x $l {expr {$x**2}}]
```

lmap s'utilise aussi avec plusieurs listes :

```
set lx [list 1 2 3 4 5 6 7 8 9]
set ly [list 9 8 7 6 5 4]
set lc [lmap x $lx y $ly {expr {$x**2}} {expr {$y**3}}]
```

Remarquons que si les listes ne sont pas de la même longueur, des éléments vides seront créés pour la compléter la plus courte.

Une astuce pour trouver les éléments non-communs à deux listes (différence symétrique) :

```
set l [lmap x $lx {if {$x in $ly} continue}]
```

3.6.2 Les dictionnaires

Il y a deux nouveautés à ajouter aux dictionnaires. Ce sont **dict map** et **dict filter**.

A l'instar de **lmap**, la sous-commande **dict map** réalise le même type d'opérations pour les dictionnaires :

```
set d [dict create a 1 b 2 c 3 d 4 e AAA]
set r [dict map {key value} $d {
   try {
      expr {$value**2}
   } on error {} {
      continue
   }
}]
```

Dans l'exemple ci-dessus, on évite les valeurs non-numériques en gérant les exceptions.

Quant à la sous-commande **dict filter**, elle retourne un sous-ensemble d'un dictionnaire selon une règle de filtrage des données. Le filtrage peut porter sur la clef, la valeur ou les deux.

```
# Création d'un dictionnaire avec des types de fichier
set type [dict create .jpg "image" .jpeg "image" .html "texte" .txt
"texte" mp3 "son" .gif "image" .mp4 "video"]
```

Filtre sur la valeur :

```
set flt1 [dict filter $type value {ima*}]
```

Filtre sur la clef :

```
set flt2 [dict filter $type key {.jp*}]
```

Filtre sur la clef ou sur la valeur :

```
set flt3 [dict filter $type script {key value} {expr {$key eq "mp4"
|| $value eq "video"}}]
```

3.7 Les canaux

3.7.1 Fermeture des canaux bidirectionnels

La commande **close** permet de fermer un canal correspondant à l'ouverture d'un fichier ou d'un socket. Dans le cas où le canal est bidirectionnel, on peut le fermer en écriture (w) ou en lecture (r).

La syntaxe est la suivante :

```
close canal r | w
```

Sans les options r | w le canal est fermé totalement.

3.7.2 Les canaux virtuels

La commande **chan push** permet d'attacher une transformation particulière à un canal.

La syntaxe est la suivante :

```
chan push canal [transformation]
```

Où transformation est une procédure à écrire prenant en charge la transformation du flux de données qui transitera par le canal.

A titre d'exemple on pourrait crypter la sortie standard en faisant :

```
chan push stdout [cryptage]
```

Et pour revenir à l'état précédent :

```
chan pop stdout
```

On peut empiler plusieurs transformations avec **chan push** (stacked channel). La commande **chan pop** dépilera alors la dernière transformation. C'est à dire celle qui se trouve au sommet de la pile. Si aucune transformation n'est présente, cela revient à fermer le canal.

Avant la version 8.6, ces fonctionnalités n'étaient disponibles qu'avec l'API du langage C (Tcl_StackChannel, Tcl_UnstackChannel, Tcl_GetStackedChannel, Tcl_GetTopChannel).

3.8 Autres nouveautés de Tcl 8.6

- Des commandes acceptent zéro arguments sans broncher : **file delete**, **file mkdir**, **glob**, **global**, **lassign**, **linsert**, **lrepeat**, **namespace upvar**, **tcl::tm::path add**, **tcl::tm::path remove** et **variable**
- A partir de Tcl 8.6, la version avec thread devient la version par défaut. Plus besoin de la compiler spécialement.

Le support du protocol IPV6 pour être à la page :

- La gestion des fichiers temporaires avec **file tempfile**,
- Un contrôle plus fin du chargement des bibliothèques avec la commande **load -global** et **-lazy**,
- Une nouvelle variable intéressante si vous faites de la programmation multiplate-forme : **tcl_platform(pathSeparator)** détermine le caractère qui sert de séparateur dans le **PATH**.

(La liste de toutes les variables s'obtient en faisant **parray tcl_platform**).

Notons aussi une dizaine de nouveautés concernant l'API C qui n'ont pas été abordées dans cet article.

4 Les nouveautés de Tk 8.6

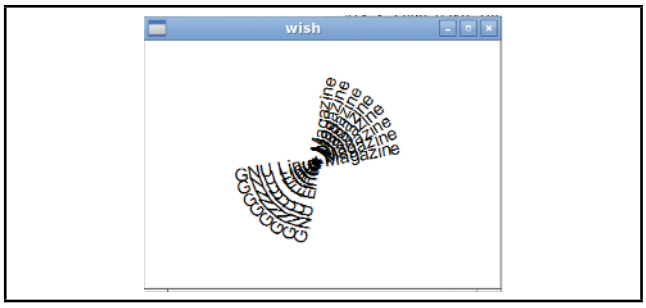
Les nouveautés sont plus modestes pour le toolkit Tk sachant que la majeure partie du travail de cette version 8.6 s'est concentrée sur Tcl.

4.1 Les nouvelles options du widget canvas

4.1.1 Le texte en oblique

Le widget **canvas** (conteneur d'objets graphiques) autorise l'affichage de texte en oblique selon un angle en degré :

```
pack [canvas .c -bg white]
foreach theta {10 20 30 40 50 60 70 80} {
      .c create text 140 140 -text "GNU Linux Magazine" -font {Arial
14} -angle $theta
}
```



4.1.2 Le déplacement d'objets graphiques

Le déplacement absolu d'objets dans le widget **canvas** se fait avec la commande **canvas moveto**.

Les objets du **canvas** sont déplacés par rapport à l'origine du repère orthonormal (coin en haut à gauche) et non pas par rapport aux dernières coordonnées de l'objet (ce que fait **canvas move**).

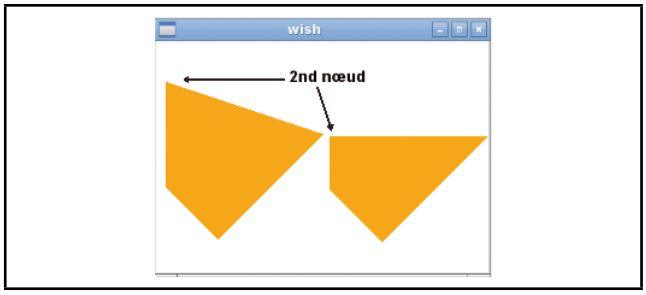
Déplacement en diagonal d'un carré orange de 60x60 pixels :

```
pack [canvas .c]
set carre [.c create rect 20 20 80 80 -fill orange]
for {set i 20} {$i <= 120} {incr i 2} {.c moveto $carre $i $i}
```

Le déplacement de noeuds avec **canvas imove** (espérons que le terme **imove** n'ait pas été déposé par la firme à la pomme !).

Dans cet exemple avec **canvas imove**, on déplace le 2nd nœud d'un quadrilatère orange qui passe des coordonnées (60,180) à (60,120). La numérotation des nœuds va de 0 à n-1.

```
pack [canvas .c]
set poly [.c create polygon 60 60 60 180 120 240 240 120 -fill orange]
.c imove $poly 1 60 120
```



4.1.3 La substitution de texte ou de noeuds

La sous-commande **canvas rchars** permet de substituer du texte pour un objet **text** ou de substituer des coordonnées pour un objet graphique possédant des nœuds (**line, polygon**).

Dans ce premier exemple, on substitue les trois lettres du mot GNU au mot NEW. L'indice des lettres variant de 0 à n-1.

```
pack [canvas .c]
set text [.c create text 140 140 -text "GNU Linux Magazine" -font {Arial 14}]
.c rchars $text 0 2 NEW
```

Dans ce second exemple, on modifie les coordonnées du troisième nœud (180,120) dans une courbe de Bézier :

```
pack [canvas .c]
set bz [.c create line 60 180 120 60 180 120 240 60 300 180 -smooth 1]
.c rchars $bz 4 5 {180 180}
```

Une courbe de Bézier s'obtient à partir d'un objet **line** ou **polygon** auquel on applique l'option -**smooth**.

4.2 Tk busy pour geler une fenêtre

La commande **tk busy** permet de suspendre l'interactivité d'une fenêtre si besoin durant un traitement. Ce qui signifie que le clavier, les boutons et le curseur de la souris sont inactifs sur la fenêtre en question. Le curseur peut être remplacé par un sablier ou une montre.

Pour neutraliser tous les widgets d'une fenêtre, il suffit d'appliquer **tk busy** au widget le plus haut dans la hiérarchie :

```
frame .top
entry .top.e1
button .top.b1 -text Run -command Run
button .top.b2 -text Exit
pack .top.e1 .top.b1 .top.b2
pack .top

proc Run {} {
 tk busy hold .top ;# Suspension de l'interactivité
 update
 tk busy configure .top -cursor "watch"
 ...
 # Phase de traitement
 ...
 tk busy forget .top ;# On redonne la main
}
```

4.3 Le format d'image PNG et les couleurs du Web

Tk supporte nativement le format PNG en lecture/écriture avec le contrôle de la transparence (canal alpha).

Pour afficher une image au format PNG avec une transparence de 0.6 :

```
set photo [image create photo -file monimage.png -format "png -alpha 0.6"]
pack [label .l -image $photo]
```

Les noms des couleurs dans Tk ont été redéfinis pour suivre les standards du Web (html,CSS) au lieu de ceux de X11. Ainsi les couleurs gray/grey, green, maroon et purple auront un rendu plus sombre **[CLR]**.

De même, les couleurs aqua, crimson, fuchsia, indigo, lime, olive, silver et teal ont été ajoutées à la liste des noms de couleurs reconnus.

4.4 Autres nouveautés de Tk

- Un nouveau sélecteur de fonte **tk fontchooser** qui fournit une interface portable au sélecteur de fonte de la plate-forme, qu'il soit modal ou non.
- La modernisation des menus déroulants en cascade sous X11.
- Un changement de la gestion de l'événement qui est actif dès que la souris est sur une fenêtre.

5 Nouveautés plus générales

5.1 Tcllib et Tklib les deux bibliothèques standards

Les deux bibliothèques standards, Tcllib **[TCLLIB]** et Tklib **[TKLIB]**, continuent de s'étoffer fortes respectivement de 400 packages répartis dans 110 modules et de 75 packages répartis dans 27 modules. Entrer dans le détail nécessiterait un autre article mais sachez que ces packages couvrent la cryptographie, les mathématiques, différents protocoles de communication, des utilitaires de texte ou de fichiers, des structures de données complexes, la représentation graphique de données, de diagrammes et de tableaux.

Par exemple :

Une fonction de hashage.

```
% package require sha256
% sha2::sha256 "GNU Linux Magazine"
% 2dbfbd7d9588ce0bf0035b7414f879921b0c282bb390352e69e213531ee5072c
```

L'algorithme phonétique d'indexation soundex.

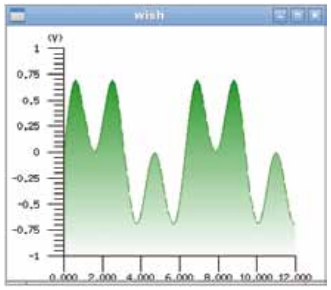
```
% package require soundex
% foreach s {concert cancer conforté confronté} {puts
[::soundex::knuth $s]}
C526
C526
C516
C516
```

Le tracé d'un graphique 2D.

```
package require Plotchart
pack [canvas .c -bg white -width 400 -height 320]
set pi 3.1415926
set p [::Plotchart::createXYPlot .c {0 12 2} {-1 1 0.25}]
$p background gradient green top-down
$p dataconfig voltage -filled up -fillcolour white -color green
$p ytext "(V)"
```

```
$p xconfig -ticklength 20 -format %.3f
$p yconfig -ticklength 20 -minorticks 4 -labeloffset 10

for {set i 0} {$i < 100} {incr i} {
   set phase [expr {2.0*$pi*$i/50.0}]
   $p plot voltage $phase [expr {0.9*cos($phase)*sin(2*$phase)}]
}
```



5.2 Les commandes dissimulées du namespace tcl::unsupported

Le namespace **tcl::unsupported** contient des commandes expérimentales susceptibles d'être intégrées dans une prochaine version de Tcl.

tcl::unsupported::assemble :

Comme pour la plupart des langages dynamiques, Tcl compile les programmes dans un code intermédiaire (bytecode) qui est ensuite exécuté par l'interpréteur. Il est donc envisageable d'écrire du code Tcl directement dans l'assembleur du bytecode qui gère une pile de donnée.

Un exemple avec ce code classique qui calcule les nombres de Fibonacci :

```
Vproc tcl::mathfunc::fib n {expr {$n < 2 ? $n : fib($n-1) + fib($n-2)}}
```

Ce qui nous donne dans l'assembleur du bytecode :

```
proc fib n {
 ::tcl::unsupported::assemble {
       load n   ;# n
       dup   ;# n n
       push 1   ;# n n 1
       gt  ;# n n>1
       jumpFalse done ;# n

       push 1   ;# n 1
       sub   ;# n-1
       push fib ;# n-1 fib
       dup   ;# n-1 fib fib
       over 2   ;# n-1 fib fib n-1
       invokeStk 2 ;# n-1 fib fib(n-1)

       reverse 3  ;# fib(n-1) fib n-1
       push 1   ;# fib(n-1) fib n-1 1
       sub   ;# fib(n-1) fib n-2
       invokeStk 2 ;# fib(n-1) fib(n-2)

       add   ;# fib(n)
       label done ;# result
 }
}
```

L'état de la pile est décrite après le symbole **#** en commentaire. Certaines mnémoniques rappellent les instructions du langage Forth ou Postscript.

Le gain en vitesse est de l'ordre de 30 % à 40 %. Mais c'est encore très expérimental ! Ce qui signifie que l'on n'est pas à l'abri d'un plantage de la machine virtuelle !

tcl::unsupported::disassemble : Il désassemble le bytecode des commandes, des expressions lambda, des scripts, des méthodes ou des objets :

```
proc add {a b} {
 return [expr {$a + $b}]
}
```

```
% tcl::unsupported::disassemble proc add
ByteCode 0x011CA980, refCt 1, epoch 5, interp 0x00835EA0 (epoch 5)
Source "\n return [expr {$a + $b}]\n"
Cmds 2, src 26, inst 7, litObjs 0, aux 0, stkDepth 2, code/src 0.00
Proc 0x011CE4C0, refCt 1, args 2, compiled locals 2
slot 0, scalar, arg, "a"
slot 1, scalar, arg, "b" Commands 2:
1: pc 0-5, src 2-24 2: pc 0-4, src 10-23
Command 1: "return [expr {$a + $b}]"
Command 2: "expr {$a + $b}"
[0] loadScalar1 %v0 # var "a"
[2] loadScalar1 %v1 # var "b"
[4] add
[5] done
[6] done
```

Le code désassemblé ne correspond pas pour le moment au code de l'assembleur !

tcl::unsupported::representation : Il décrit la représentation interne d'une donnée :

```
% tcl::unsupported::representation 7
value is a pure string with a refcount of 3, object pointer at
011FB620, string representation "7".
% set a "Linux"
% tcl::unsupported::representation $a
value is a pure string with a refcount of 4, object pointer at
011F7A40, string representation "Linux".
% tcl::unsupported::representation [list 2 4 8 16]
value is a list with a refcount of 1, object pointer at 0104E588,
internal representation 01205350:00000000, no string representation.
```

tcl::unsupported::inject : Cette commande permet d'injecter un code dans une **coroutine** pendant qu'elle est suspendue à des fins de débogage par exemple.

5.3 Migration de l'infrastructure de développement

Le développement de Tcl/Tk migre de CVS vers Fossil **[FOSSIL]**, un logiciel de gestion de version développé par D. Richard Hipp (le créateur de SQLite).

Par exemple, pour cloner le code source de Tcl-Tk :

```
% fossil clone http://core.tcl.tk/tcl/ tcl.fossil
% fossil clone http://core.tcl.tk/tk/ tk.fossil
```

6 Tcl/Tk 9 pour la suite ?

Eh bien la suite sera Tcl/Tk 9 dont le développement a déjà commencé depuis le début de l'année. A l'instar de Perl 6 ou Python 3, il faut s'attendre à une rupture de la rétro-compatibilité. Surtout au niveau de l'API du langage C. Cependant elle ne sera pas radicale car cela reste un facteur important pour les développeurs. Mais ce ne sera plus le facteur le plus important.

Il faut s'attendre également à une sérieuse amélioration du rendu de Tk dont l'aspect ne correspond plus tellement aux canons esthétiques actuels. Tk tiendra d'autant plus compte du côté tactile des récentes interfaces graphiques.

Conclusion

Tcl/Tk continu d'évoluer à son rythme et s'apprête à fêter dignement ses 25 ans en 2013. On peut s'étonner du rythme très espacé des versions. Mais après un quart de siècle (ce qui est un temps très long en informatique!), on aspire à plus de stabilité.

Le principal intérêt pour migrer vers Tcl/Tk 8.6 est l'officialisation de la POO. Mais au-delà, les coroutines nous font envisager d'autres niveaux de programmation.

Ce langage stable bénéficie toujours d'une base d'utilisateurs et de contributeurs fidèles tant auprès des développeurs professionnels que des amateurs.

Pour en savoir plus dans le détail sur toutes ces nouveautés, n'oubliez pas de lire le manuel **[MAN]**. ■

Références

[CLR] http://en.wikipedia.org/wiki/X11_color_names#Color_name_clashes

[DODECA] http://www.tcl.tk/man/tcl/TclCmd/Tcl.htm

[FOSSIL] http://www.fossil-scm.org/

[MAN] http://www.tcl.tk/man/tcl8.6/

[STARKIT] Découvrez Tclkit, les starkits et les starpacks, GNU Linux Magazine France n° 60, Avril 2004, pp66-69.

[TCLLIB] http://core.tcl.tk/tcllib/home

[TKLIB] http://core.tcl.tk/tklib/home

[ZLIB] http://www.zlib.net/

CRÉER UNE APPLICATION TIZEN DE APP A ZEN

par Philippe « RzR » Coval [Ingénieur Eurogiciel Rennes] & Théo Guérin [Etudiant – ESIR Rennes]

Pour ceux qui n'auraient pas suivi l'actualité récente, Tizen est la plate-forme mobile GNU/Linux HTML5 à venir. Laissez vous guider par ce tutoriel qui s'adresse à tout niveau de développeur et dont le but est de vous faire acquérir les connaissances nécessaires à la création d'une application Tizen.

1 Introduction

1.1 Contexte

Cet article se veut comme un partage d'expérience suite au développement d'une application mobile Tizen chez mon employeur (Eurogiciel). De l'installation de l'environnement à la publication d'une application sur le Tizen Store en passant par son développement, le chemin est long et parsemé d'embûches mais la réalisation n'en est que plus gratifiante.

Avant de commencer, retenez le point d'entrée du projet Tizen pour les développeurs **[0]**, très important pour récupérer les bonnes informations. Les outils de communication associés sont à votre disposition : un wiki, une liste de diffusion, un forum, un chat...

1.2 Historique

Le système GNU/Linux sur les mobiles n'est pas vraiment une nouveauté puisqu'il remonte à plus d'une dizaine d'années. L'histoire est assez longue mais on retiendra le projet Openmoko, pour sa grande ouverture, et Maemo de Nokia, pour son aboutissement et son succès commercial.

Courant 2010, les deux géants Intel et Nokia décident d'un projet commun appelé MeeGo afin de mutualiser les efforts de distributions GNU/Linux des deux compagnies : respectivement Moblin et Mameo. La valeur ajoutée des deux partenaires étant principalement la vente de matériels, il y a donc fort à parier qu'un échange collaboratif est préférable à un échange compétitif.

C'est alors que le 11 février 2011, Microsoft se rapproche de Nokia (ou l'inverse) qui s'oriente de plus en plus vers Windows Phone avant même d'en obtenir l'exclusivité. Cependant, cette nouvelle n'empêchera pas la sortie du Nokia N9, un produit très compétitif mais sans avenir commercial. Intel se retrouve alors seul à porter le projet et recherche des partenaires. Samsung, ayant déjà le modèle Limo basé sur la plate-forme Linux, manifeste son intérêt pour une coopération. Et nous voilà repartis pour un tour, le temps de faire un peu de rebranding et de se détacher de l'association avec le précèdent partenaire Nokia.

Tizen devient le nom de cette nouvelle aventure qui poursuit la voie ouverte par le projet Meego. La boucle est bouclée, et voici une nouvelle ère de terminaux mobiles GNU/Linux. Un bref historique ne faisant jamais de mal, reste à savoir ce qu'est Tizen. Tizen est une plate-forme ouverte, un nouveau système d'exploitation innovant basé sur le standard HTML5. Que ce soit sur un smartphone, une tablette, une smart TV ou dans un véhicule en tant que dispositif embarqué, Tizen est un système capable de s'adapter à tout type de matériel.

Ce projet open source, reposant sur la puissance du noyau Linux, est soutenu par la Tizen Association. Cette association regroupe de nombreuses organisations, constructeurs, opérateurs et industriels supportant le projet, tels que The Linux Foundation, Intel ou Samsung. Étant fondé sur HTML5, Tizen est orienté vers les technologies web mais pas seulement. Une application peut être développée dans un environnement aussi bien web avec HTML5 que natif en C. De plus, ces deux frameworks peuvent même être combiné dans une application hybride où l'on peut jauger cet équilibre selon ses préférences. Un développeur pourra ainsi privilégier soit une grande simplicité et réutilisabilité avec le web et HTML5, soit une meilleure performance en natif.

2 Description

2.1 Architecture

Détailler le coeur de l'architecture Tizen **[1]** est un peu fastidieux pour démarrer, mais il reste important de savoir que le système, basé sur un système entièrement GNU/Linux,

propose un Web RunTime. Il est notamment composé de libc, la bilbliothèque standard de C, et non d'une machine virtuelle Java comme Android.

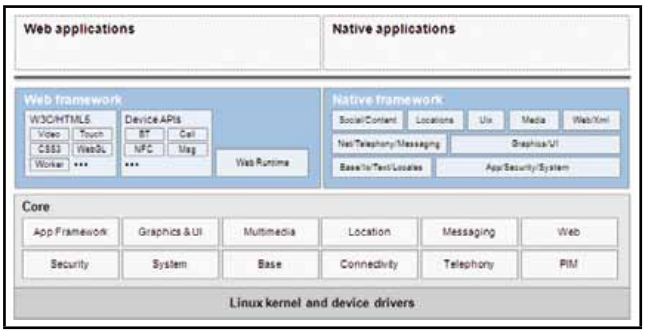


Fig. 1

Reposer sur le standard HTML5 est un gros point fort pour Tizen. En effet, HTML5 est devenu une référence dans le développement web grâce au travail du W3C. Lorsque le terme HTML5 est utilisé dans cet article, il englobe aussi bien HTML5 que CSS3 et JavaScript.

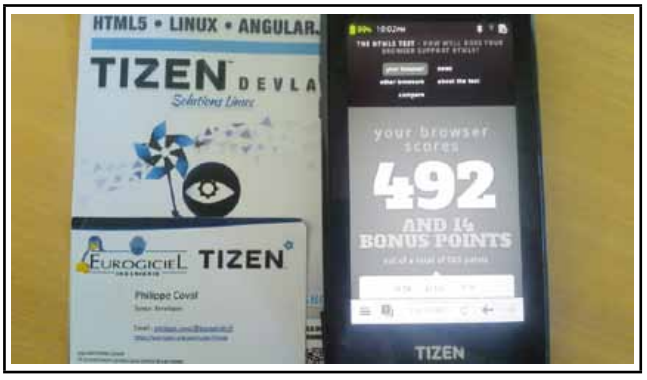


Fig. 2

Au delà de toutes les API HTML5 standardisées, Tizen peut profiter d'API spécifiques permettant d'accéder aux possibilités qu'offre le matériel. Tizen ne réinvente pas la roue et c'est pourquoi, les API HTML5 et Tizen cohabitent harmonieusement afin d'exploiter le meilleur de chaque technologie.

Au final, le développeur dispose de technologies modernes, aussi bien pour la mise en forme avec CSS3 que pour la création applicative avec JavaScript. Vous aurez sans doute l'occasion d'utiliser l'une des bibliothèques JavaScript efficaces tel que jQuery Mobile ou jqMobi. De nombreuses API sont à la disposition du développeur : Geolocalisation, Web Storage, File API, Canvas, Web Sockets, Drag & Drop...

2.2 Modèle de sécurité

Les smartphones prennent de plus en plus de place dans notre vie et par conséquent de plus en plus d'informations personnelles sur les utilisateurs y sont stockées. Par cette évolution, la sécurité est devenue une réelle préoccupation dans le domaine de la téléphonie mobile. Voilà pourquoi Tizen doit remplir son rôle de protection du système et des données privées de l'utilisateur.

La plate-forme accepte donc cette responsabilité de contrôle et cherche à informer l'utilisateur sans pour autant limiter la variété d'applications disponible. Pour se faire, le système oblige l'application à demander un privilège afin de pouvoir utiliser des informations ou des fonctionnalités précises. Ainsi, l'utilisateur est informé sur ce que l'application qu'il s'apprête à installer peut ou ne peut pas faire. Accéder à ses contacts, son calendrier, ses fichiers ou sa position sera alors impossible sans demander à l'utilisateur. Il pourra alors décider en son âme et conscience de faire confiance en l'application selon sa provenance.

La gestion de ces privilèges est gérée dans les métadonnées du projet : **`config.xml`** pour les applications web et **`manifest.xml`** pour les applications natives. Ces privilèges sont classés en trois niveaux différents en fonction de son impact potentiel sur le système :

- Les privilèges **publics**, ouverts à tous développeurs.
- Les privilèges des **partenaires**, ne pouvant être utilisés que par les développeurs identifiés en tant que partenaires du Tizen Store.
- Les privilèges liés à la **plate-forme**, utilisés dans les API système pour gérer la plate-forme Tizen et seulement ouvert à un ensemble spécifique de développeurs d'applications Tizen.

De plus, l'application doit être liée à un profil de sécurité déterminé par deux types de signatures :

- La signature de l'auteur. Elle indique l'identité de l'auteur et garantie l'intégrité de l'application confirmée par le développeur grâce à un certificat qu'il doit générer.
- La signature du distributeur. Elle est générée par le publieur d'application, notamment le Tizen Store. Celui-ci détermine le niveau de privilège dans lequel l'application se situe, ce qui permettra ou non l'utilisation de certains privilèges.

La vérification est faite de bout en bout, de la génération de paquetage à l'installation de ce dernier. Voilà ce que l'utilisateur peut observer au démarrage d'une application demandant certains privilèges : (Fig. 3).

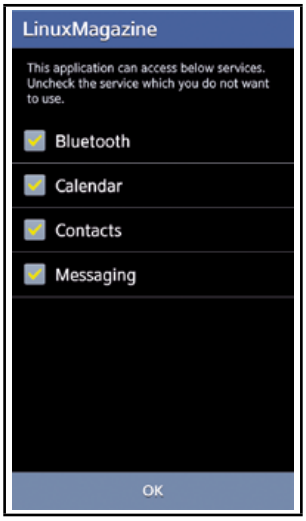


Fig. 3

3 Environnement

3.1 Installation SDK

Le SDK basé sur Eclipse est multiplate-forme, mais pas forcément multi-distribution. Donc voici la procédure sur la plate-forme GNU/Linux de référence, Ubuntu. Vous pouvez toujours l'installer dans un chroot ou l'isoler dans une machine virtuelle préalablement préparée. Cependant, il vous faudra faire attention à la virtualisation pour l'affichage 3D du simulateur basé QEmu.

Une fois sa distribution installée, on installe quelques dépendances :

```
sudo apt-get install \
time gettext expect libudev-dev \
qemu-user-static libwebkitgtk-1.0-0
```

Par contre il faudra installer la version Java d'Oracle, soit par l'intermédiaire de l'Ubuntu Software Center, soit via repo tiers PPA **[1.5]** :

```
sudo apt-get remove openjdk-7-jre openjdk-6-jre-lib openjdk-7-jre-headless
default-jre
sudo tar xvfz ~/Downloads/jdk-7u25-linux-*.tar.gz -C /usr/local/opt/
sudo ln -fs /usr/local/opt/jdk1.7.0_25 /usr/local/jdk
```

Cette version sera utilisée, il faut dont l'ajouter dans l'environnement (**~/.bashrc**) du développeur :

```
export PATH=${PATH}:/usr/local/opt/jdk1.7.0_25/bin
```

Une vérification de la version Java s'impose :

```
java -version # vérifier qu'il pointe pas sur openjdk
Java(TM) SE Runtime Environment (build 1.7.0_25-b15)
```

On est donc prêt pour télécharger la dernière version de Tizen SDK sur https://www.tizen.org/fr et lancer le téléchargement :

```
chmod +x tizen-sdk-*.bin
time ./tizen-sdk-*.bin
```

Profitez du temps de téléchargement assez long pour lire les chapitres suivants... Puis lancer l'IDE eclipse :

```
~/tizen-sdk/ide/eclipse
```

Voir Figure 4.

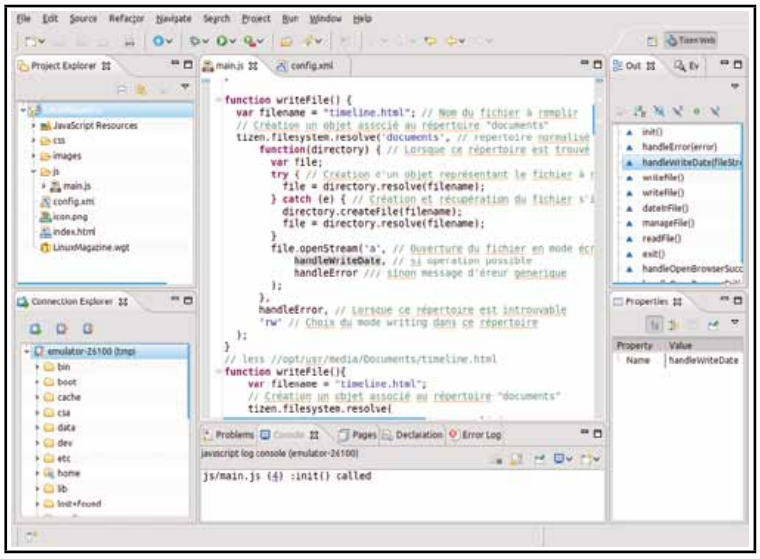


Fig 4 : Vue de l'interface d'Eclipse

3.2 Usage du SDK

Inutile de décrire Eclipse par énumération de toutes ses options, l'outil a été conçus pour être plutôt intuitif (si si!). Voici tout de même la procédure pour créer une application Hello World basique. Nous nous servirons de cette base pour tester certaines API dans les chapitres suivants.

Prise en main :

- IDE: **~/tizen-sdk/ide/eclipse**
- Menu : File > New > Tizen Web Project
- Tizen > Template > Basic > Blank Application > Project name : « HelloWorld »

Squelette projet :

- Point d'entrée : **index.html**
- Configuration : **config.xml**
- Logique : **main.js**
- Apparence : ***.css *.png**

Web Simulator :

- Simuler le Web Runtime (WRT) :
- Rapide mais peu réaliste (performance et RT du système hôte)
- Basé sur WebKit (Google Chrome)
- Évènements simulables :
  - → Appels Téléphoniques
  - → Hardware : Sensor, GPS, PM, NFC...
  - → Usage : IDE > Run As > Tizen Web Simulator application

Emulateur :

- Emulation : VM firmware target Tizen (x86)
- Plus fidèle que la simulation mais moins performant

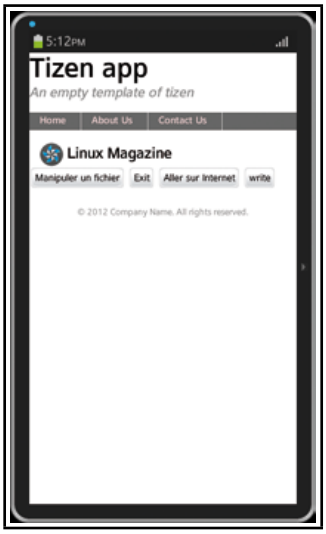


Fig. 5

- Connection Explorer : liste RT (dev, emu)
- Exporte le système fichiers de chaque cible
- Usage :
  → IDE : Window > Show View > Connection Explorer : (>)
  → Emulator Manager : tizen > x86-s > create new : Create ; Launch
  → IDE : Run As > Web Application (Fig. 5).

4 Développement

4.1 API FileSystem

Voici un exemple de l'API Application, un simple appel de fonction suffit pour terminer le cycle de vie de l'application :

```
tizen.application.getCurrentApplication().exit();
```

On peut directement l'invoquer via un événement bouton dans le fichier « index.html »

```
<button onclick="tizen.application.getCurrentApplication().exit();">
Quitter
</button>
```

Autre exemple, une grande nouveauté HTML5 est l'API File System. Celle-ci nous permet facilement de manipuler un fichier. Créer, écrire, lire ou simplement extraire de l'information, tout ceci est possible en seulement quelques lignes de code. **[4]**

Avant toute chose il faut ajouter les permissions désirées dans le fichier **config.xml** du projet :

```
<tizen:privilege name="http://tizen.org/privilege/filesystem.write"/>
<tizen:privilege name="http://tizen.org/privilege/filesystem.read"/>
```

Concernant le code, cela se passe dans le fichier **js/main.js** généré par l'IDE :

```
function handleError(error) {
  console.log("error: " + error.message );
}

function handleWriteDate(fileStream) {
  // Ajoute la date courante dans le fichier
  fileStream.write("\r"+new Date());
  fileStream.close();
}

function writeFile() {

  var filename = "timeline.html"; // Nom du fichier ou ecrire

  // Création un objet associé au répertoire logique "documents"
  tizen.filesystem.resolve
    (
      'documents', // 1er argument: repertoire normalisé

      function(directory) { // 2eme : Lorsque ce répertoire est trouvé
        var file;
        try { // obtention d'un descripteur de fichier
          file = directory.resolve(filename);
        } catch (e) { // sinon création et récupération du fichier
          directory.createFile(filename);
          file = directory.resolve(filename);
        }
        file.openStream('a', // Ouverture du fichier en mode ajout
            handleWriteDate, // si operation possible
            handleError /// sinon message d'erreur générique
        );
      },

      handleError, // 3eme : Lorsque ce répertoire est introuvable

      'rw' // 4eme :  Choix du mode writing dans ce répertoire

  );
}
```

Cette fonction peut être lancée par un événement **click** sur un bouton, mais aussi dans une fonction d'initialisation. Apres exécution, on peut simplement vérifier le contenu du ficher dans le file manager ou se connecter à la cible via le Developement bridge **sdb** :

```
~/tizen-sdk/tools/sdb shell
less //opt/usr/media/Documents/timeline.html
```

On peut maintenant adapter ce code pour faire une opération symétrique de lecture en utilisant l'API W3C **FileReader**. Pour cela on remplace le paramètre **handleWriteDate** de **openStream** par la fonction anonyme suivante :

```
function()  {

  // Lecture du fichier
  file.readAsText(

    // Affichage de la taille et du contenu du fichier
    function(contents) {
      console.log('File size: ' + file.fileSize);
      console.log('File contents: ' + contents);
    },

    function(e) {
      console.log('Could not read the file: ' + e.message);
    }

  );
}
```

Pour en savoir plus, je vous propose d'aller voir le sample **FileManager**.

4.2 API SystemInfo

On peut aussi afficher l'état de la batterie en se basant sur le précédent exemple **FileWriter** :

```
function(fileStream) {

  tizen.systeminfo.getPropertyValue
    ( "BATTERY", // Récupération de la propriété de la batterie

      function(battery) {
        fileStream.write("level:"+battery.level);
```

```
        fileStream.close();
    },

    handleError
  );
}
```

4.3 API Application

Voici une utilisation de cette API Application un peu plus complexe où le navigateur Web est lancé dynamiquement au sein de l'application.

On ajoute le privilège dans **config.xml** et le code dans le source JavaScript **js/main.js** :

```
<tizen:privilege name="http://tizen.org/privilege/application.launch"/>


function handleOpenBrowserSuccess() {
  console.log("launch internet application control succeed");
}

function openBrowser() {
   // url cible : le fichier crée précédemment ou tout type d'url
   var url="file:///opt/usr/media/Documents/timeline.html";

   // Créer un objet ApplicationControl
   // où l'on définit les fonctionnalités requises de l'application
   var appControl = new tizen.ApplicationControl
      ( "http://tizen.org/appcontrol/operation/view", // Opération désirée
        url);

   // Cherche et lance l'application qui répond aux critères
   tizen.application.launchAppControl
     ( appControl,
       null,
       handleOpenBrowserSuccess,
       handleError
     );

}
```

4.4 Publication

La phase de publication ne comporte pas de difficulté particulière, il suffit d'uploader l'application en .wgt après s'être connecté sur : http://seller.tizenstore.com/. Il ne nous reste alors plus qu'à ajouter une description et des photos d'écran à la taille demandée. Réaliser une photo est d'une facilité déconcertante grâce à l'émulateur : Clic droit sur l'émulateur > Advanced > Screen Shot. Ensuite, voici une astuce image magick pour redimensionner l'image dans la taille demandée :

```
convert -resize '480x854!' screenshot.png '480x854.tmp.png'
```

5 Support Hardware

Pour les chanceux qui ont un accès à un matériel de référence Tizen RD-210, ce chapitre est fait pour vous. Pour les autres, vous devez vous contenter de la combinaison émulateur+simulateur expliquée plus haut. Quand aux vrais téléphones Tizen, les rumeurs fusent depuis l'annonce du projet. Certains opérateurs et constructeurs parleraient d'une sortie pour la fin de l'année 2013.

Fig. 6

5.1 Procédure de flashing du device RD-210 sous tizen-2.2 :

Le premier support de l'OS est un modèle de référence fournis par Samsung lors de la Tizen conf 2012. RD-210 était fournit avec la version **larkspur** (deb) qui est désormais obsolète [NDLR : le RD-210 basé sur le Galaxy S2 avec SoC Exynos 4210 (C210) est maintenant obsolète et remplacé par le RD-PQ basé sur le Galaxy S3 et utilisant un SoC Exynos 4412 (PQ)]

5.1.1 Bootloader

On enlève la batterie et le câble USB, on presse les boutons vol+ [NDLR : ou vol-] et power, et l'écran **uboot** apparaît. On peut alors brancher l'USB et uploader les images (Fig. 6).

5.1.2 Flashing

On flash en 3 étapes pour être sur une bonne base :

```
PATH="${HOME}/tizen-sdk/tools:${PATH}" && export PATH
url="http://download.tizen.org/releases/system/Tizen_Disk_Migration.tar" # 1s
wget -c "$url"
sudo time lthor "Tizen_Disk_Migration.tar" # 1s

url="http://download.tizen.org/releases/system/Tizen_Ref.Device-210_
System_20130207_1.tar" # 4s
wget -c "$url"
sudo time lthor "" # 1s

url="http://download.tizen.org/snapshots/2.2/common/latest/images/RD-210/
tizen-2.2_20130807.3_RD-210.tar.gz"
wget -c "$url"
sudo time lthor "" # 1s
```

5.1.3 Reboot

Le système reboote et en plus de l'affichage on peut se logger dans le device :

```
$ sdb shell "cat /proc/version"
Linux version 3.0.15-00002-g277d62a ...

$ sdb shell "cat /etc/os-release"
NAME="Tizen"
VERSION="2.2.0, (Tizen)"
ID=tizen
PRETTY_NAME="Tizen 2.2.0 (Tizen)"
VERSION_ID="2.2.0"
```

Abonnez-vous !

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

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

11 Numéros de GNU/Linux Magazine

60€*

au lieu de **86,90 €*** en kiosque

Économie : **26,90 €***

Économisez plus de **30%***

* Sur le prix de vente unitaire France Métropolitaine

Nouveauté 2014
TOUS LES HORS-SÉRIES PASSENT EN GUIDE !

Un nouveau format avec une reliure de luxe pour ces Guides de référence de 128 pages à conserver dans votre bibliothèque !

Découvrez tous les Guides déjà parus sur

boutique.ed-diamond.com

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

Les 3 bonnes raisons de vous abonner :

- Ne manquez plus aucun numéro.
- Recevez GNU/Linux Magazine chaque mois chez vous ou dans votre entreprise.
- Économisez 26,90 €/an ! (soit plus de 3 magazines offerts !)

4 façons de commander facilement :

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

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

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

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

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

❑ Je souhaite recevoir les offres promotionnelles de nos partenaires.

GNU LINUX MAGAZINE / FRANCE

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

Vos remarques :

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

Tournez SVP pour découvrir toutes les offres d'abonnement

PROFITEZ DE NOS OFFRES D'ABONNEMENT SPÉCIALES !

➡ Tous les abonnements incluant GNU/Linux Magazine :

offre LM

INCLUS : GNU/Linux Magazine (11nos)

60€* au lieu de 86,90€** en kiosque

Économie 26,90€

offre LM+

INCLUS : GNU/Linux Magazine (11nos) + ses 6 Guides

115€* au lieu de 164,30€** en kiosque

Économie 49,30€

NOUVEAUTÉ 2014 TOUS LES HORS-SÉRIES PASSENT EN **GUIDE !**

offre T

198€* au lieu de 277,10€** en kiosque

Économie 79,10€

INCLUS : GNU/Linux Magazine (11nos), Open Silicium (4nos), MISC (6nos), Linux Pratique (6nos) et Linux Essentiel (6nos)

offre T+

+6 GUIDES **+3 GUIDES** **+2 Hors-Séries**

299€* au lieu de 411,20€** en kiosque

Économie 112,20€

INCLUS : GNU/Linux Magazine (11nos) + ses 6 Guides, Linux Pratique (6nos) + ses 3 Guides, MISC (6nos) + ses 2 Hors-Séries, Open Silicium (4nos) et Linux Essentiel (6nos)

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

Pour consulter l'ensemble de nos offres rendez-vous sur :

boutique.ed-diamond.com

Vous pouvez également commander par Tél. : +33 (0)3 67 10 00 20 / Fax : +33 (0)3 67 10 00 21

➡ Nos Tarifs s'entendent TTC et en euros		F	OM1	OM2	Zone 1	Zone 2
		France Métro	Outre-Mer		Europe	Reste du Monde
LM	Abonnement GLMF	60 €	75 €	96 €	83 €	90 €
LM+	Abonnement GLMF + 6 Guides	115 €	147 €	190 €	160 €	173 €
T	Abonnement GLMF + MISC + OS + LP + LE	198 €	275 €	325 €	276 €	300 €
T+	Abonnement GLMF + GLMF HS (6 Guides) + MISC + MISC HS + OS + LP + LP HS (3 Guides) + LE	299 €	390 €	491 €	415 €	448 €

• OM1 : Guadeloupe, Guyane française, Martinique, Réunion, St Pierre et Miquelon, Mayotte

• OM2 : Nouvelle Calédonie, Polynésie française, Wallis et Futuna, Terres Australes et Antarctiques françaises

Mes choix :

Mon 1er choix	Je sélectionne l'offre choisie :	
Mon 2ème choix	Je sélectionne l'offre choisie :	
Mon 3ème choix	Je sélectionne l'offre choisie :	
	Je sélectionne ma zone géographique :	
	J'indique la somme due : (Total)	€

Je choisis de régler par :

❑ Chèque bancaire ou postal à l'ordre des Éditions Diamond

❑ Carte bancaire n° |_|_|_|_| |_|_|_|_| |_|_|_|_| |_|_|_|_|

Expire le : |_|_|_|_|

Cryptogramme visuel : |_|_|_|

Date et signature obligatoire

On ajoute le support 3D hardware (non libre) :

```
$ x-www browser http://source.tizen.org/mali-ddk-2.2

$ unp ~/Downloads/mali_2.2.tar.gz
$ sdb push . /tmp/   # on uploade les rpm

$ sdb shell # on se connecte sur le systeme cible pour les installer

$ su # on passe root sur la cible \o/
# zypper mr --disable Tizen-main # inactive le repo non dispo
# zypper install  -n -f /tmp/*.rpm # 1,y,R (on force l'install des rpms)
# reboot
```

5.2 Configuration

Ensuite on génère et uploade les certificats : IDE > Window > Preferences > Tizen SDK > Native > Security Profiles > Generate. Puis on peut lancer l'application grâce au Connection Explorer, comme pour l'émulateur. Au passage on peut configurer la connection wifi pour ajouter les repos publics (à utiliser avec précaution) :

```
$ sdb shell
$ su
# zypper ar http://download.tizen.org/live/Tizen/standard/Tizen.repo
# zypper ref
# zypper install xterm screen # ... et autres utilitaires
```

A noter que notre device n'a pas les boutons hardware « Back » et « Menu » apparus dans la version tizen 2.2. Pour éviter de rester coincé, il faut parfois quitter l'application en cours comme les Settings en pressant longuement le bouton « Home » et reprendre le déroulement des écrans pour arriver a la bonne page.

Si vous voulez rester sur la dernière version officiellement supporté par ce device, adaptez l'opération avec la version 2.1 : http://download.tizen.org/releases/2.1/tizen-2.1/images/RD-210/tizen-2.1_20130517.6_RD-210.tar.gz

Conclusion

L'objectif de cet article était de vous présenter Tizen de façon générale. Cependant, il y aurait encore beaucoup à dire car chaque point aurait pu faire l'objet d'un article indépendant. Pour vous familiariser avec le développement pour la plate-forme Tizen, je vous conseille de jeter un coup d'oeil aux codes souvent simples des démos dans le SDK Tizen. Vous pourrez ainsi mieux comprendre comment utiliser les API, qu'elles soient simplement HTML5 ou propres à Tizen. Vous aurez aussi un premier exemple des configurations de sécurité à apporter à votre application.

Bien sûr, un système d'exploitation n'est rien sans matériel. Dans les prochains mois, nous en sauront plus sur les premières sorties Tizen des constructeurs. Les plus attendues sont bien évidemment les smartphones dont certains constructeurs auraient annoncé la sortie dès cet automne. Supportant les API natives C++, Tizen aurait alors l'avantage d'être compatible avec les applications du premier système mobile de Samsung, Bada. Le système mobile pourrait donc arriver sur le marché avec une base d'applications non négligeable pour son intégration dans le marché.

Le système Linux Tizen est décliné sous différentes formes et ne se limite pas seulement aux applications JavaScript. En effet le core du système possède un fort potentiel car certaines instances Tizen supportent d'autres technologies (On peut citer EFL, C++, OSP/Bada, Qt, GTK, GNU...). On garde a l'esprit que les produits finaux Tizen auront probablement différents degrés d'ouverture. Cependant, il faut retenir qu'ils devront tous permettre de développer des applications en langage JavaScript avec des bibliothèques JavaScript comme jQuery Mobile, jqMobi, AngularJS. Les frameworks web multiplate-formes, comme Apache Cordova/ Phonegap, aussi sont une piste intéressante ayant un avenir assuré. En effet, ces technologies sont d'un intérêt évident pour les développeurs, n'ayant qu'à développer une application une seule fois pour qu'elle soit valable sur n'importe quel système.

Il reste encore beaucoup à dire sur un sujet large comme Tizen car c'est un projet énorme et que l'on peut confronter ou comparer à d'autres technologies comme FireFoxOS, BB10 WebWorks. Chaque point que nous avons vu ensemble pourrait encore être développé, cela fera peut-être l'objet d'un prochain article spécialisé dans un domaine précis. Dans cet article, nous avons vu l'ensemble du processus de création d'une application Tizen , de la configuration à la publication. Il ne vous reste plus qu'à laisser libre cours à votre créativité pour réaliser et partager vos idées les plus folles ! ■

Remarque

[0] https://developer.tizen.org/

[1] https://dockr.eurogiciel.fr/blogs/embedded/solution-linux-2013-tizen-architecture-et-sdk/

[1.5] http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

[2] https://wiki.tizen.org/wiki/Install_Tizen_SDK_on_Ubuntu

[3] http://www.youtube.com/watch?v=WsW8psRlW6U

[4] https://developer.tizen.org/fr/documentation/articles/html5-features-on-tizen

[5] http://www.orangepartner.com/articles/interview-mark-sage-tizen-new-contender

ON A PERDU LE MINITEL MAIS ON A TOUJOURS L'ANNUAIRE...

LDAP : 1 - LE CÔTÉ SERVEUR

par Cédric PELLERIN [Utilisateur de GNU/Linux depuis 1993]

L'accès centralisé aux informations est un graal que de nombreuses générations d'administrateurs système ont rêvé d'atteindre. Diverses solutions sont apparues sur le marché mais la plus répandue actuellement s'intitule Lightweight Directory Access Protocol ou encore LDAP pour les intimes. Nous allons essayer de lever un coin du voile sur ce mystérieux et ô combien effrayant service.

Lorsque que les sociétés ont commencé leur grande migration du mainframe vers les serveurs « micro », les besoins n'étaient pas du tout les mêmes que maintenant. A l'époque, posséder 3-4 serveurs (un par service majeur en gros) sur lesquels on posait les fichiers et qui pilotaient la laser partagée et éventuellement le fax était déjà un luxe. A cette époque, un utilisateur se connectait à un seul serveur et il y trouvait tout ce dont il avait besoin. Les administrateurs système se contentaient donc de créer le compte sur un serveur et le tour était joué.

Cependant, dès 1988 certains se sont posés la question du futur et du comment centraliser les comptes, les adresses e-mail voire les adresses physiques, le numéro de bureau ou le nom de l'assistante ainsi que sa photo. Ces réflexions ont abouti à une spécification nommée X.500 dont sont inspirés, mais seulement inspirés, les systèmes d'annuaire actuels. Sur cette base sont arrivés successivement :

- Netware Directory Services (NDS) apparu en 1993 avec la version 4 de Novell Netware
- LDAP lui aussi dans les années 1993 à l'université du Michigan dont le dérivé libre OpenLDAP a fait son apparition en 1998
- Active Directory implémenté dans Windows NT 4 server en 1999

et de très nombreux autres dont le dernier, mais pas le moindre, est Samba 4 avec son LDAP intégré afin de simuler un Active Directory.

1 Introduction

Mais au fait, qu'est-ce que cette fameuse norme X.500 dont on parle depuis tout à l'heure ? « X.500 désigne l'ensemble des normes informatiques sur les services d'annuaire définies par l'UIT-T (anciennement appelé CCITT) » (source Wikipedia). Il s'agit donc d'un annuaire, version électronique du gros livre qui servait à retrouver le numéro de téléphone de la grande-tante une fois par an pour lui présenter nos vœux avant l'arrivée du Minitel et d'internet. Cet annuaire possède une caractéristique essentielle qu'il ne faut pas perdre de vue, il est organisé hiérarchiquement. L'image la plus proche que nous connaissons bien est celle de l'organigramme d'entreprise. En haut nous avons le PDG, en dessous nous avons le DRH, le DAF et le DT, puis sous le DT nous avons toute l'organisation technique de la société avec le SI (DSI[1], ingénieurs, techniciens, etc.) et ainsi de suite. Une arborescence X.500 (ou LDAP) est en général très proche de cela. Attention cependant, aucune norme ou recommandation officielle n'impose quoi que ce soit en ce qui concerne l'arborescence. C'est à vous et à vous seul de la définir, de la maintenir et d'en assurer le support. Certains logiciels clients partent de suppositions qu'il va falloir transformer à grands coups d'expressions régulières rageuses.

Un annuaire OpenLDAP est constitué d'un grand nombre d'objets définis dans des fichiers schémas. Ces objets sont organisés hiérarchiquement entre eux (un peu comme les classes en programmation objets). Ils sont nommés objectClass et possèdent des attributs nommé attributes. Ces objectClasses et leurs attributes sont utilisés dans des entries lors de la création du contenu de l'annuaire. Par exemple si l'on veut créer l'organisation Tartempion qui

[1] Pour les réfractaires aux sigles, voici la traduction : DRH=Directeur des Ressources Humaines, DAF = Directeur Administratif et Financier, DT = Directeur Technique, DSI = Direction du Système d'Information (l'informatique interne)

possède le domaine **tartempion.com**, nous allons écrire l'entry suivante :

```
# Organization for Tartempion Corporation
        dn: dc=tartempion,dc=com
        objectClass: dcObject
        objectClass: organization
        dc: tartempion
        o: Tartempion Corporation
        description: The Tartempion Corporation
```

Dans cette entry nous retrouvons, outre son Distinguished Name (**dn:**), deux objectClasses (**dcObject** et **organization**) et trois attributes (**dc**, **o** et **description**).

Les notions principales à retenir pour comprendre OpenLDAP sont les suivantes :

- Concernant les schémas :
  - → Tous les objectClasses et attributes sont définis dans des schémas. Certains nommés « operational » sont définis dans le code du serveur lui-même mais il n'est pas vraiment utile de s'en préoccuper pour le moment.
  - → Tous les schémas qui définissent les objectClasses et les attributes que nous voulons utiliser doivent être connus du serveur. Par exemple si nous voulons que Samba puisse gérer son authentification via LDAP, nous devons inclure le schéma **samba.schema**.
  - → Un attribut défini dans un schéma peut être utilisé par un objectClass défini dans un autre schéma. Dans ce cas attention à l'ordre d'inclusion, il peut être important.
- Concernant les objectClasses :
  - → Les objectClasses peuvent être (et en général elles le sont) organisées hiérarchiquement auquel cas elles peuvent hériter des propriétés de leurs parents (notés SUP dans les schémas).
  - → Les objectClasses peuvent être de type :
    - « Structural » et dans ce cas elles sont utilisées pour créer des entries.
    - « Auxiliary » qui peuvent être ajoutées à une entry.
    - « Abstract » qui désigne des objets virtuels. Le plus représentatif est l'objet « top », sommet de l'annuaire et extrémité de toute hiérarchie.
  - → Si un objectClass fait partie d'une hiérarchie, elle doit être du même type que son parent, sauf si son parent est de type « abstract ».
  - → Les objectClasses sont le seul moyen d'inclure des attributs et elles définissent si ces attributs sont obligatoires ou optionnels.
  - → Les objectClasses sont définies dans les fichiers **.schemas** en utilisant la notation ASN.1 (cf. http://www.zytrax.com/books/ldap/apa/oid.html pour de plus amples explications).
- Concernant les attributes :
  - → Tout attribute est inclus dans un ou plusieurs objectClasses.
  - → Les attributes sont définis en utilisant la notation ASN.1.
  - → Un attribute peut apparaître une seule ou plusieurs fois dans son objectClass.
  - → Une définition d'attribute peut être partie intégrante d'une hiérarchie auquel cas il hérite de toutes les propriétés de ses parents.
  - → La définition d'un attribute inclut sa syntaxe et comment il réagit sous certaines conditions. Par exemple si les opérations de comparaison seront sensibles à la casse ou non.
- Concernant les entries :
  - → Une entry doit contenir une et une seule objectClass de type « Structural ».
  - → Une entry peut contenir un nombre quelconque d'objectClasses de type « Auxiliary ».
  - → Une entry peut contenir une et une seule objectClass de type « Abstract ».
  - → Les entries peuvent avoir des enfants qui apparaissent donc en dessous d'elles dans la hiérarchie.
  - → Les entries peuvent avoir des parents qui apparaissent donc au dessus d'elles dans la hiérarchie.
  - → Les entries peuvent avoir des frères et sœurs qui apparaissent donc au même niveau qu'elles dans la hiérarchie.

Reprenons à la lumière de ces informations l'entry définie plus haut :

```
# Organization for Tartempion Corporation
        dn: dc=tartempion,dc=com
        objectClass: dcObject
        objectClass: organization
        dc: tartempion
        o: Tartempion Corporation
        description: The Tartempion Corporation
```

et regardons les objectClasses :

- « organization » est définie dans **core.schema** et est de type « Structural » :

```
objectclass ( 2.5.6.4 NAME 'organization'
        DESC 'RFC2256: an organization'
        SUP top STRUCTURAL
        MUST o
        MAY ( userPassword $ searchGuide $ seeAlso $ businessCategory $
                x121Address $ registeredAddress $ destinationIndicator $
                preferredDeliveryMethod $ telexNumber $ teletexTerminalIdentifier $
```

```
            telephoneNumber $ internationaliSDNNumber $
            facsimileTelephoneNumber $ street $ postOfficeBox $ postalCode $
            postalAddress $ physicalDeliveryOfficeName $ st $ l $ description
) )
```

notre première condition pour un objet est remplie. Cette classe exige (MUST) un attribute **o** et en autorise (MAY) un paquet d'autres en option, dont **description** que nous avons ci-dessus.

- **dcObject** est, elle aussi, définie dans **core.schema** et est de type « Auxilliary ». Elle apporte le support de l'attribute **dc** qui est obligatoire :

```
objectclass ( 1.3.6.1.4.1.1466.344 NAME 'dcObject'
       DESC 'RFC2247: domain component object'
       SUP top AUXILIARY MUST dc )
```

Vous avez remarqué la suite de nombres séparés par des points et bien sûr vous vous êtes dis « Mais c'est comme une MIB SNMP ce truc ». Eh oui, la notation ASN.1 est utilisée aussi par le SNMP. Là encore ces suites de nombres nommées OID ont des significations particulières qu'il n'est pas très utile de connaître à moins de vouloir créer ses propres schémas. Vous pourrez trouver quelques précisions ici : http://www.openldap.org/doc/admin24/schema.html, ici : http://linuxgazette.net/130/peterson.html, là : http://www.yolinux.com/TUTORIALS/LinuxTutorialLDAP-DefineObjectsAndAttributes.html et ailleurs.

Dans la vraie vie, pas celle dans laquelle tuer des monstres rapporte des points de mana, quelques schémas sont utilisés presque tout le temps dès lors que l'on veut monter un serveur qui puisse aussi servir à authentifier des utilisateurs. Il s'agit de :

- **core.schema** qui regroupe les objectClasses et attributes nécessaires à la constitution d'un annuaire de base.
- **cosine.schema** qui étend **core.schema** en rajoutant des attributes et objectClasses provenant directement de X.500. Ces objets sont nettement orientés vie courante (DN of manager, home telephone number, DN of secretary, Photo, etc.), gestion de DNS (A record, MX record, etc.).
- **inetorgperson.schema** qui étend **core.schema** et **cosine.schema** et rajoute encore des détails comme le numéro de département, la plaque d'immatriculation de la voiture et autres.
- **nis.schema** qui, comme son nom l'indique, reprend les entrées du feu Network Information Service (aka Yellow Pages) qui fut aussi un système de centralisation de comptes[2].

Parmi tous ces schémas, il faut retenir certains objectClasses et attributes simples :

- **o** : organization
- **ou** : organizationalUnit
- **cn** : commonName
- **dc** : domainComponent

Un nom complet d'objet ou Distinguished Name (en abrégé « dn ») est constitué d'un ou plusieurs noms d'objets séparés par une virgule. Par exemple le domaine **test.glmf** sera noté **dn:dc=test,dc=glmf**. Le serveur **toto.test.glmf** lui-même sera nommé **dn:cn=toto,dc=test,dc=glmf**. On aurait parfaitement pu choisir autre chose qu'une appellation basée sur le nom de domaine et séparer en plusieurs entités en fonction de l'organigramme de la société. Par exemple, si **toto** est un serveur de la compta : **dn:cn=toto,ou=compta,o=glmf**.

Le plus déroutant quand on aborde LDAP c'est que l'on fait ce qu'on veut et ça peut vite donner l'impression d'un immense bor^Wchantier. Il n'y a aucune norme qui va vous dire dans tel cas utiliser plutôt les noms de domaines avec les objets **dc** et dans tel autre cas utiliser plutôt les appellations « commerciales » du type **ou=,o=**. C'est à vous de décider mais il faudra ensuite vous y tenir ou tout recommencer à zéro.

Faites très attention aux outils de migration plus ou moins automatiques que vous pourrez trouver sur le net. Ils font leur travail plutôt pas mal mais ils vont vous imposer une hiérarchie qui ne vous conviendra peut-être pas. Il vaut nettement mieux commencer par définir cette hiérarchie calmement, la valider à blanc puis prendre le temps d'écrire votre propre script de migration dans votre langage favori afin de coller à vos desiderata.

2 Le côté serveur : le daemon slapd

2.1 Analyse fonctionnelle

Depuis plusieurs articles nous sommes sur un réseau qui s'intitule **test.glmf** et qui comprend déjà un serveur DNS/DHCP, un serveur mail, un serveur Samba et un serveur Apache. La machine que nous venons de rajouter répondra au doux nom de **tyr.test.glmf**[3].

En ce qui concerne l'arborescence OpenLDAP nous allons très logiquement recopier et étendre ce réseau et ainsi créer les entrées suivantes :

- dn de base : **dc=test,dc=glmf**
- Administrateur LDAP : **cn=admin,dc=test,dc=glmf**

[2] Pour les veinards qui n'ont pas connu ça : http://en.wikipedia.org/wiki/Network_Information_Service

[3] Tyr étant le dieu des serments et du droit (dixit Wikipedia) utiliser son nom m'a parut assez à propos.

- Nos utilisateurs seront dans : **ou=people,dc=test,dc=glmf**
- Nos groupes seront dans : **ou=group,dc=test,dc=glmf**
- L'utilisateur toto par exemple : **cn=toto,ou=people,dc=test,dc=glmf**

Afin d'être tranquille nous allons charger les principaux schémas :

- **core.schema**
- **cosine.schema**
- **inetorgperson.schema**
- **openldap.schema**
- **nis.schema**

Nous allons utiliser la base de données Berkeley DB ou **bdb** qui est la plus ancienne mais aussi la plus simple à mettre en place. D'autres « backends » existent comme **hdb**, MySQL, etc. et rien ne vous empêche de faire des tests si vous voulez vous amuser.

2.2 Installation

Commençons donc par installer OpenLDAP. Pour cela nous devons prendre une décision, supporter le protocole d'encryption TLS ou non. En effet, si on veut le supporter, ce qui est franchement mieux pour la sécurité, il faut disposer d'une version supérieure ou égale à la 2.4.33. Or la plus récente version packagée chez Debian (ou Ubuntu) est la 2.4.31 et celle-ci segfault dès qu'on lui configure le support TLS. En revanche il serait faux de croire que l'on ne peut pas sécuriser ces versions. En effet, à défaut de TLS elles supportent très bien SSL via le protocole ldaps. Cependant dans le but d'être aussi exhaustif que possible, nous allons partir sur la version 2.4.35 à récupérer sur le site d'OpenLDAP et la compiler à la main. Nous allons donc exceptionnellement utiliser une version non packagée pour cet article. Si vous décidez malgré tout d'utiliser la version packagée et donc de ne pas utiliser l'encryption TLS il vous suffira d'adapter les divers chemins indiqués, adaptation qui consistera la plupart du temps à enlever le **/usr/local**. Cependant certaines subtilités existent et un encadré vous expliquera ce qu'il faut faire plus précisément.

La première chose à faire consiste à récupérer la dernière version via un classique :

```
# wget ftp://ftp.openldap.org/pub/OpenLDAP/openldap-release/
openldap-2.4.35.tgz
```

puis installons les dépendances dont nous allons avoir besoin pour compiler (en plus du kit de développement avec **make**, **gcc**, **autoconf**, etc.) :

```
# apt-get install libtool libltdl-dev libssl-dev libdb5.1-dev libsasl-dev
```

Décompactons les sources :

```
# tar zxvf openldap-2.4.35.tgz
```

puis lançons la compilation :

```
# cd openldap-2.4.35
# ./configure --enable-crypt=yes --enable-lmpasswd=yes --enable-
spasswd=yes --enable-modules=yes --enable-overlays=yes
# make depend
# make
# make install
```

Normalement tout devrait bien se passer et après ce « certain temps » si cher au regretté Fernand Raynaud[4] on devrait obtenir notre binaire serveur nommé **slapd** dans **/usr/local/libexec** et les outils répartis entre **/usr/local/bin** et **/usr/local/sbin**.

Afin de pouvoir faire tourner le serveur autrement qu'avec les droits **root**, nous allons créer un utilisateur spécifique nommé **openldap**. Cet utilisateur n'aura pas de **shell** et ne pourra donc pas être utilisé pour une tentative d'intrusion.

```
# useradd -s /bin/false -d /usr/local/var/openldap-data openldap
```

2.3 Configuration

2.3.1 Configuration de base du serveur

L'une des grandes nouveautés de la version d'OpenLDAP 2.4 est l'utilisation fortement recommandée de ce qu'on appelle l'OLC pour On-Line Configuration, aussi connue sous le nom de **cn=config**. Apparu en mode test avec la version 2.3, ce système déporte la configuration du serveur du fichier **slapd.conf** vers l'arborescence LDAP elle-même. En gros, OpenLDAP contient sa configuration en lui-même. Cette approche peut paraître complexe voire injustifiée mais c'est le succès d'OpenLDAP qui est à l'origine de cette modification. En effet, quand on utilise le fichier plat **slapd.conf**, chaque modification nécessite le redémarrage du serveur, c'est à dire l'arrêt du service pendant quelques secondes, ou quelques minutes si la base est grosse et le serveur peu puissant ou chargé. Comme OpenLDAP est utilisé par de grosses entreprises et administrations comme centralisateur de comptes, il est sollicité à chaque authentification, chaque envoi de mail, chaque lecture des boîtes aux lettres, etc. Un arrêt d'une minute de ce service peut avoir des conséquences fâcheuses. Certes la duplication et la notion de serveur maître et esclave existe depuis plus de dix ans mais si l'on pouvait trouver un moyen de modifier la configuration sans redémarrage, ce serait aussi bien. C'est ce qui a été fait ici et nous pouvons reconfigurer nos serveurs à chaud via la commande **ldapmodify** et ses petites sœurs.

[4] http://www.deezer.com/en/track/13710150 pour les ignares ou les nouveaux-nés :)

Au tout début, il faut bien partir sur une configuration vide et faire quand même tourner le serveur qui va dépendre de cette configuration qui... Bref, a-t'on réinventé le problème de l'œuf et de la poule ? Pas vraiment car le fichier **slapd.conf** va nous permettre de créer la configuration initiale avant le premier démarrage du serveur. Ce fichier est assez simple à comprendre si l'on a bien saisi la notion d'arborescence de LDAP et si on a en tête celle que l'on veut utiliser. Pour coller à notre analyse fonctionnelle, voici notre fichier **slapd.conf** dans toute sa splendeur :

```
include   /usr/local/etc/openldap/schema/core.schema
include   /usr/local/etc/openldap/schema/cosine.schema
include   /usr/local/etc/openldap/schema/inetorgperson.schema
include   /usr/local/etc/openldap/schema/openldap.schema
include   /usr/local/etc/openldap/schema/nis.schema

pidfile         /usr/local/var/run/slapd.pid
argsfile        /usr/local/var/run/slapd.args

access to dn.base="" by * read
access to dn.base="cn=Subschema" by * read
access to *
        by self write
        by users read
        by anonymous auth

database config
rootdn          "cn=manager,cn=config"
rootpw          password

database        bdb
suffix           "dc=test,dc=glmf"
rootdn          "cn=admin,dc=test,dc=glmf"
rootpw          password

directory       /usr/local/var/openldap-data

index   objectClass     eq
index   uid     eq
index   cn,gn,mail eq,sub
index   ou eq
index   default eq,sub
```

Les premières lignes d'**include** permettent d'insérer les schémas que nous voulons.

Dans le paragraphe suivant nous expliquons à **slapd** où mettre certains fichiers utiles à son fonctionnement. Ensuite nous donnons les droits de base sur l'arborescence qu'il est possible de résumer ainsi :

- **access to dn.base="" by * read** : tout le monde a le droit de lire les données de base du serveur (DSE[5]) ;
- **access to dn.base="cn=Subschema" by * read** : ainsi que sa sous-arborescence ;

mais en ce qui concerne nos propres données à nous :

- **access to * by self write** : le « propriétaire » c'est à dire l'utilisateur dont le dn correspond à l'entrée en cours peut les écrire (et donc les lire). Par exemple, j'ai le droit de modifier mon mot de passe ou mon numéro de téléphone dans l'annuaire mais pas ceux des autres.
- **by users read** : un utilisateur déjà connecté peut tout lire (exemple quelle est l'adresse de courriel de Dupont ?).
- **by anonymous auth** : les autres ont le droit de s'authentifier avant toute chose.

Ensuite nous précisons que les paramètres qui suivent concernent une première base de données de type « config », la fameuse OLC ou **cn=config**. Là nous créons juste un administrateur (**rootdn**) nommé manager auquel nous affectons le mot de passe « password ».

Dans le paragraphe suivant nous précisons le type de base de données à utiliser pour notre domaine **test.glmf** qui sera bdb pour Berkely DB ainsi que le suffixe de base pour nos entrées, le nom de l'administrateur ainsi que son mot de passe, ici mis en clair pour rester simple mais il est évident que sur une machine en production il faut utiliser un hachage (S)MD5 ou (S)SHA[6]. Ceci est possible grâce à l'utilitaire **slappasswd**.

Ensuite nous signifions à OpenLDAP de mettre ses fichiers de base de données dans **/usr/local/var/openldap-data** puis nous créons quelques indexes sur les entrées de cette base de données afin d'accélérer les recherches.

Ce fichier rempli, il va nous falloir le transformer en format LDIF afin d'utiliser le nouveau système de configuration pour le serveur **slapd**[7]. Commençons par créer le répertoire qui va accueillir la configuration au nouveau format :

```
# mkdir /usr/local/etc/openldap/slapd.d
```

puis utilisons l'utilitaire **slaptest** pour effectuer la transformation :

```
# cd /usr/local/etc/openldap
# slaptest -f slapd.conf -F slapd.d
```

slaptest va se plaindre violemment qu'il ne trouve pas un fichier concernant la base de données, il suffit de l'ignorer, le travail a quand même été effectué.

Toutes ces manipulations ayant été effectuées en tant que root, nous devons donner les droits sur les fichier à l'utilisateur **openldap** pour que **slapd** puisse les lire :

```
# chown -R openldap:openldap /usr/local/etc/openldap
```

[5] DSE signifie DSA Specific Entry. DSA signifie Directory System Agent. En gros DSE désigne une entrée de contrôle interne au serveur, par exemple le contexte de nom de base (dc=test,dc=glmf pour nous) ou la version de LDAP supportée.

[6] Le S initial est là pour seed. Il s'agit de la version avec graine des deux protocoles.

[7] A noter que, pour le moment, le fichier slapd.conf est toujours pris en compte s'il existe. Pour être sûr que slapd utilise bien la nouvelle norme de configuration, vous pouvez soit utiliser un autre nom que slapd.conf soit préciser au serveur slapd quelle méthode utiliser via l'option -F suivie du chemin complet vers le répertoire slapd.d.

Une fois ceci fait, il nous reste à créer le fichier **/usr/local/var/openldap-data/DB_CONFIG** que **slapd** va utiliser pour gérer les bases de type Berkeley DB. Un exemple est fourni et installé lors du **make install**, il suffit de le renommer :

```
# mv /usr/local/var/openldap-data/DB_CONFIG.
example /usr/local/var/openldap-data/
DB_CONFIG
```

puis de donner les droits sur ce répertoire à l'utilisateur **openldap** :

```
# chown -R openldap:openldap /usr/local/var/
openldap-data
```

et nous pouvons enfin démarrer notre serveur :

```
# /usr/local/libexec/slapd -u openldap -g
openldap -h 'ldap:///'
```

Les options **-u** et **-g** indiquent sous quel utilisateur et groupe le serveur doit tourner et l'option **-h** indique le type de connexion supportée, ici pour le moment on se contente d'une connexion simple. Il est possible de rajouter la connexion notée **ldapi:///** qui ajoute le support des IPC, ce qui peut servir pour certains clients. Cela donnerait **-h 'ldap:/// ldapi:///'**.

Pour passer en mode debug et interdire au serveur de se mettre en arrière-plan, il faut rajouter l'option **-d** avec une valeur comprise entre 1 et 0xffffffff. Pour avoir la liste tapez :

```
# /usr/local/libexec/slapd -d ?
```

En règle générale la valeur 3 donne de bons résultats.

Si le serveur se lance sans problème, nous pouvons effectuer un premier test avec la commande :

```
# slapcat -s cn=config
```

qui devrait nous afficher toute la configuration trouvée. Il est fortement conseillé de « piper » cette commande dans **less** si vous souhaitez examiner le résultat. Un autre test révélateur consiste à se connecter réellement avec le **dn** de l'administrateur de la base **config**, c'est à dire **cn=manager,cn=config** :

Note

Afin de simplifier le démarrage des utilisateurs de Debian ou autres distributions similaires qui auront préférés installer les paquets **slapd** et **ldap-utils**, voici le fichier **slapd.conf** à utiliser :

```
include         /etc/ldap/schema/core.schema
include         /etc/ldap/schema/cosine.schema
include         /etc/ldap/schema/inetorgperson.schema
include         /etc/ldap/schema/openldap.schema
include         /etc/ldap/schema/nis.schema

pidfile         /var/run/slpad/slapd.pid
argsfile        /var/run/slpad/slapd.args

modulepath      /usr/lib/ldap
moduleload      back_bdb.la

access to dn.base="" by * read
access to dn.base="cn=Subschema" by * read
access to *
        by self write
        by users read
        by anonymous auth

database config
rootdn          "cn=manager,cn=config"
rootpw          password

database      bdb
suffix            "dc=test,dc=glmf"
rootdn          "cn=admin,dc=test,dc=glmf"
rootpw         password

directory       /var/lib/ldap

index   objectClass     eq
index   uid     eq
index   cn,gn,mail eq,sub
index   ou eq
index   default eq,sub
```

Dans notre étude nous avons intégré tous les backends en dur à la compilation, sous Debian ils sont en modules et il faut donc charger ceux dont on a besoin au démarrage. D'où l'ajout des deux directives **modulepath** et **moduleload**.

Vous remarquerez aussi les modifications des divers chemins d'accès pour coller aux options retenues par le mainteneur du paquet.

Attention, lors de l'installation du paquet, une configuration par défaut est créée. Il faut donc commencer par remettre tout à zéro en suivant les directives contenues dans la note suivante.

En suivant ce pas à pas, vous obtiendrez peut-être une erreur au démarrage de **slapd** du type **alock package is unstable**. Dans ce cas, il suffit de se positionner dans le répertoire hébergeant la base de données :

```
# cd /var/lib/ldap
```

et de lancer l'utilitaire **db5.1_recover**. Une fois ceci fait, il faut bien remettre **openldap** comme propriétaire sur tous ces fichiers :

```
# chown openldap:openldap *
```

puis relancer **slapd** et là il devrait démarrer sans problème.

En ce qui concerne les diverses options (**-h**, **-d**, etc.) il faut les positionner dans **/etc/default/slapd**.

```
# ldapsearch -b cn=config -D "cn=manager,cn=config" -w password
```

qui devrait donner le même résultat. Une contre-vérification :

```
# ldapsearch -b cn=config -D "cn=manager,cn=config" -w toto
ldap_bind: Invalid credentials (49)
```

2.3.2 Injection des données

Notre serveur est opérationnel mais pour le moment il ne contient rien d'autre que sa propre configuration. Il est donc temps de lui rentrer nos informations. Pour insérer des données dans un serveur LDAP, il faut obligatoirement passer par des fichiers au format LDIF. Ces fichiers ont une caractéristique très importante à bien connaître : chaque espace ou saut de ligne compte. Le mode « list » de **vim** (**:set list**) est là très utile. Un fichier LDIF est constitué d'une série d'entrées séparées par une ligne vide. Il doit aussi toujours se terminer par une ligne vide sinon la dernière entrée ne sera pas prise en compte.

Nous allons commencer par rajouter la base de notre annuaire ainsi que son administrateur. Vous me direz qu'on vient de le faire, certes, mais c'était pour la branche **cn=config**, pas pour notre arbre de données. Le fichier nommé **init.ldif** est le suivant :

```
dn: dc=test,dc=glmf
objectclass: dcObject
objectclass: organization
o: GNU Linux Magazine France
dc: test

dn: cn=admin,dc=test,dc=glmf
objectclass: organizationalRole
cn: admin
```

et il s'insère via la commande :

```
# ldapadd -x -D "cn=admin,dc=test,dc=glmf" -w password -f init.ldif
```

La commande **ldapadd** permet d'insérer des données dans un annuaire de manière douce. Son pendant violent s'appelle **slapadd** et n'est utilisable que par l'utilisateur **root** sur le serveur proprement dit alors que **ldapadd** supporte de passer par le réseau. La commande **slapadd** est à éviter sauf problème majeur.

Les paramètres de **ldapadd** ici indique que nous utilisons le mode d'authentification de base, c'est à dire un échange en clair (flag **-x**), que notre utilisateur pour nous connecter est **cn=admin,dc=test,dc=glmf** (flag **-D** pour Distinguished Name), que notre mot de passe est « password » (flag **-w**, **-W** seul pour faire en sorte que **ldapadd** le demande) et les données à insérer sont contenues dans le fichier **init.ldif**.

Si vous n'avez pas fait de faute de frappe, les deux champs devraient s'insérer sans souci. Pour le valider, suffit de demander :

```
# ldapsearch -LLL -x -D "cn=admin,dc=test,dc=glmf" -w password -b
'dc=test,dc=glmf' '(objectclass=*)'
```

oui je sais, on pouvait faire plus compliqué mais c'était plus cher !

Nous pouvons maintenant faire la même chose avec les **OU** (*Organizational Units*) de base qui nous serviront à créer les utilisateurs et les groupes. Là encore vous pouvez choisir ce que vous voulez comme noms, LDAP n'est qu'un annuaire, vous y mettez ce que vous voulez. Pour la clarté de cet article nous avons choisi d'appeler l'**OU** des utilisateurs « people » et l'**OU** des groupes « groups ». Il faut donc créer le fichier **ou.ldif** suivant :

```
dn: ou=people,dc=test,dc=glmf
objectClass: organizationalUnit
ou: people

dn: ou=groups,dc=test,dc=glmf
objectClass: organizationalUnit
ou: groups
```

et l'insérer avec la même commande que ci-dessus :

```
# ldapadd -x -D "cn=admin,dc=test,dc=glmf" -w password -f ou.ldif
```

Pour les utilisateurs les champs à remplir sont un peu plus nombreux. Par exemple pour créer un utilisateur **cpellerin**, le fichier **users.ldif** sera semblable à celui-ci :

```
dn: cn=cpellerin,ou=people,dc=test,dc=glmf
objectClass: top
objectClass: account
objectClass: posixAccount
objectClass: shadowAccount
uid: cpellerin
uidNumber: 1000
gidNumber: 10000
userPassword: password
gecos: Cedric Pellerin
loginShell: /bin/bash
homeDirectory: /home/cedric
shadowWarning: 7
shadowMin: 8
shadowMax: 9999
shadowLastChange: 10877
```

où l'on reconnaît les principaux champs habituels ainsi que quelques autres liés aux schémas utilisés. Par exemple les champs **shadow*** sont définis par la RFC 2307 et on peut trouver leur définition à l'url suivante : http://docs.oracle.com/cd/E19513-01/806-4251-10/mapping.htm

Certains de ces attributs sont obligatoires, d'autres non. Pour le savoir, il faut aller rechercher comment est défini l'objectClass dans le fichier schéma correspondant grâce à l'utilitaire qui va bien : **grep**[8]. La syntaxe des fichiers schema

[8] Si quelqu'un a une autre méthode plus pratique, je suis preneur...

a déjà été définie plus tôt mais nous allons voir ici un exemple concret. Prenons par exemple l'objectClass **shadowAccount**, sa définition dans le fichier **nis.schema** est :

```
objectclass ( 1.3.6.1.1.1.2.1 NAME 'shadowAccount'
        DESC 'Additional attributes for shadow passwords'
        SUP top AUXILIARY
        MUST uid
        MAY ( userPassword $ shadowLastChange $ shadowMin $
              shadowMax $ shadowWarning $ shadowInactive $
              shadowExpire $ shadowFlag $ description ) )
```

ce qui signifie que pour s'intégrer dans cette classe d'objets le champs **uid** est obligatoire (clause MUST) alors que les autres sont optionnels (clause MAY), le signe **$** servant de séparateur.

Nous venons d'affecter à l'utilisateur **cpellerin** le groupe de **gid** 10000. Il serait temps de le créer grâce au fichier **groups.ldif** ci-dessous :

```
dn: cn=ldap,ou=groups,dc=test,dc=glmf
objectClass: top
objectClass: posixGroup
cn: ldap
gidNumber: 10000
```

Vous noterez au passage que LDAP n'étant qu'un annuaire, il ne fera aucune validation sur la cohérence de vos données. Rien ne vous empêche d'affecter quinze fois le même UID à quinze utilisateurs différents ni de leur affecter des GID correspondants à des groupes inexistants.

Un petit coup de **ldapadd** plus tard et notre groupe est inséré. Nous pouvons nous prendre pour un utilitaire de login et demander à ldap quel shell est associé à l'utilisateur **cpellerin** :

```
# ldapsearch -x -D 'cn=cpellerin,ou=people,dc=test,dc=glmf' -w
password -b 'ou=people,dc=test,dc=glmf' '(cn=cpellerin)' loginShell
```

ce qui devrait nous retourner la réponse suivante :

```
# extended LDIF
#
# LDAPv3
# base <ou=people,dc=test,dc=glmf> with scope subtree
# filter: (cn=cpellerin)
# requesting: loginShell
#

# cpellerin, people, test.glmf
dn: cn=cpellerin,ou=people,dc=test,dc=glmf
loginShell: /bin/bash

# search result
search: 2
result: 0 Success

# numResponses: 2
# numEntries: 1
```

Ici nous avons testé que nous pouvons nous connecter avec le compte d'un utilisateur lambda et récupérer correctement un paramètre de son compte. Donc tout va bien.

Nous avons maintenant un serveur OpenLDAP opérationnel mais un peu dangereux à utiliser même dans un réseau local car tout passe en clair sur le réseau en ce moment. Il est donc temps de passer à la vitesse supérieure et de sécuriser tout ça.

Note **Remise à zéro de la configuration.**

Si à la suite de certaines manœuvres plus ou moins orthodoxes vous souhaitez remettre à zéro la configuration du serveur, voici la procédure à suivre :

arrêter le serveur :

```
# killall slapd
```

supprimer la configuration de base :

```
# rm -rf /usr/local/etc/openldap/slapd.d/*
```

Faire vos modifications dans **slapd.conf**

Recréer la configuration au format LDIF et donner les droits :

```
# cd /usr/local/etc/openldap
# slaptest -f slapd.conf -F slapd.d
# chown -R openldap:openldap /usr/local/etc/openldap
```

Si vous voulez juste modifier la configuration de base sans effacer les données, vous vous arrêtez là et vous relancez **slapd**.

Si, en plus, vous désirez supprimer les données, il faut aller purger la base **bdb** en sauvegardant le fichier **DB_CONFIG** :

```
# mv /usr/local/var/openldap-data/DB_CONFIG ~
# rm -rf /usr/local/var/openldap-data/*
```

remettre le fichier **DB_CONFIG** à sa place et donner le bon propriétaire aux fichiers :

```
# cp ~/DB_CONFIG /usr/local/var/openldap-data/
# chown -R openldap:openldap /usr/local/var/openldap-data
```

redémarrer le serveur.

2.4 Sécurisation

2.4.1 Sécuriser la communication grâce à TLS

TLS (pour Transport Layer Security) est un protocole de sécurité bien connu des administrateurs systèmes qui ont déjà installés leur serveur de messagerie. Pour les autres le FM à lire se situe ici : http://en.wikipedia.org/wiki/Transport_Layer_Security. En effet il sert beaucoup dans les communications POP3S et IMAPS. Nous allons l'utiliser aussi ici pour sécuriser l'accès à notre OpenLDAP. Pour ce faire

nous devons commencer par créer une clé de chiffrage et ensuite la signer avec notre propre autorité de certification (auto-signature). Commençons par la clé RSA 2048 bits :

```
# openssl genrsa -out private.pem 2048
```

puis créons le certificat auto-signé correspondant :

```
# openssl req -new -x509 -key private.pem -out cert.pem -days 1095
```

Lors de l'exécution de cette commande **openssl** va vous poser un tas de questions indiscrètes sur votre lieu de vie, votre entreprise, l'âge de votre poisson rouge, etc. Vous pouvez répondre ce que vous voulez SAUF à la question « Common Name » à laquelle vous devez répondre par le fqdn complet de votre serveur si vous voulez vous éviter des problèmes à la connexion. Si vous n'avez pas d'entrée DNS correspondant à votre machine, je vous suggère très fortement d'aller mettre son IP et son FQDN dans le fichier **/etc/hosts**.

Ceci effectué, il reste à copier le fichier **private.pem** dans **/usr/local/etc/openldap**[9] et le fichier **cert.pem** dans **/etc/ssl/certs** en le renommant avec le nom du serveur (**tyr** ici). Ensuite il faut créer un lien symbolique, toujours dans **/etc/ssl/certs** avec le hash de **tyr.pem** :

```
# cd /etc/ssl/certs
# ln -s tyr.pem `openssl x509 -noout -hash -in /etc/ssl/certs/tyr.pem`
```

ce qui devrait vous donner un lien du genre :

```
5fd6f667 -> /etc/ssl/certs/tyr.pem
```

Il nous faut maintenant expliquer à OpenLDAP comment utiliser ces fichiers nouvellement créés. Ajoutons donc les lignes suivantes à la fin du fichier **/usr/local/etc/openldap/slapd.conf** :

```
security ssf=128
TLSCipherSuite HIGH:MEDIUM:+TLSv1:!SSLv2:+SSLv3
TLSCACertificateFile /etc/ssl/certs/tyr.pem
TLSCertificateFile /etc/ssl/certs/tyr.pem
TLSCertificateKeyFile /usr/local/etc/openldap/private.pem
TLSVerifyClient never
```

puis refaire la procédure de conversion avec **slaptest** après avoir arrêté **slapd** bien entendu.

Après cela, il nous reste à modifier la partie « cliente », c'est à dire le fichier **/usr/local/etc/ldap.conf** en ajoutant les lignes :

```
ssl     start_tls
tls_cacert      /etc/ssl/certs/tyr.pem
```

Enfin nous pouvons relancer OpenLDAP, vérifier qu'il ne râle pas et essayer de nous connecter :

```
# ldapsearch -h tyr.test.glmf -ZZ -D "cn=admin,dc=test,dc=glmf" -w
password -b 'dc=test,dc=glmf' '(objectclass=*)'
```

Ici l'option **-ZZ** force une connexion TLS valide, c'est à dire notamment que le CN renseigné dans le certificat correspond bien au fqdn de votre serveur. L'option **-Z** seule demande le cryptage mais sans vérifier la validité du certificat.

Si lors du debug vous voyez passer l'erreur **openldap main: TLS init def ctx failed: -1** cela signifie que **slapd** ne peux pas lire la clé privée. Vérifiez bien les droits d'accès.

2.4.2 Mais moi je veux du SSL

L'usage du protocole TLS n'est pas exclusif et peut être très facilement combiné avec du SSL. Cela aura pour effet de rajouter un serveur sécurisé écoutant sur un autre port. Pour obtenir cela à partir d'un serveur déjà configuré pour le support TLS, il suffit de relancer **slapd** en lui spécifiant un protocole supplémentaire via l'option **-h**[10] :

```
# /usr/local/libexec/slapd -u openldap -g openldap -h 'ldap:/// ldaps:///'
```

et ainsi le faire écouter en non chiffré et en chiffré TLS sur le port 389 et en chiffré SSL sur le port 636.

Pour ceux qui utilisent une version bugguée ou qui ne veulent pas de TLS, il faut aller mettre dans **slapd.conf** uniquement les lignes suivantes :

```
TLSCACertificateFile /etc/ssl/certs/tyr.pem
TLSCertificateFile /etc/ssl/certs/tyr.pem
TLSCertificateKeyFile /usr/local/etc/openldap/private.pem
```

Ces directives expliquent au serveur où aller chercher les clés et certificats nécessaires.

Une fois le serveur relancé un **netstat -nlp** devrait nous montrer un serveur **slapd** écoutant sur le port 636. Avant de continuer, il faut maintenant expliquer aux applications clientes où aller chercher le certificat nécessaire à l'encryption. Cela se fait dans le fichier **ldap.conf** sis dans **/usr/local/etc/openldap** (ou **/etc/ldap** pour les versions packagées Debian-like). Il faut y rajouter la ligne suivante :

```
TLS_CACERT      /etc/ssl/certs/tyr.pem
```

Ensuite, pour valider son fonctionnement, notre vieil ami **ldapsearch** doit subir une légère modification de syntaxe :

```
# ldapsearch -H ldaps://tyr.test.glmf/ -x -D 'cn=cpellerin,ou=people,dc=test,dc=
glmf' -w password -b 'ou=people,dc=test,dc=glmf' '(cn=cpellerin)' loginshell
```

où l'option -H force le serveur à utiliser ainsi que le protocole.

2.4.3 Un petit coup de SASL

Tant que nous y sommes, autant aussi chiffrer la partie authentification sur le serveur LDAP. Pour ce faire nous allons utiliser une méthode assez simple nommée SASL pour

[9] On peut aussi le mettre dans /etc/ssl/private en le renommant tyr-key.pem par exemple. Dans ce cas il faut ajouter l'utilisateur openldap au groupe ssl-cert pour que slapd puisse lire la clé.

[10] Pour les utilisateurs d'une version packagée, cela se fait aussi dans /etc/default/slapd sous l'option SLAPD_SERVICES

Simple Authentication and Security Layer[11] qui ici sera basée sur un échange de clés. Il serait possible de relier tout ça à un serveur Kerberos mais là franchement j'ai autre chose à faire. Nous allons donc nous contenter d'une authentification de type DIGEST-MD5 en utilisant le serveur SASL de Cyrus. Il faut rajouter quelques paquets sur notre serveur :

```
# apt-get install sasl2-bin libsasl2-modules
```

et rajouter quelques lignes à **slapd.conf** :

```
sasl-realm openldap
sasl-host tyr.test.glmf
sasl-secprops none
authz-regexp uid=admin@openldap,cn=openldap,cn=DIGEST-MD5,cn=auth
cn=admin,dc=test,dc=glmf
authz-regexp uid=(.*)@openldap,cn=openldap,cn=DIGEST-MD5,cn=auth
cn=$1,ou=people,dc=test,dc=glmf
```

Nous définissons ici le royaume SASL, nommé de façon fort originale **openldap**, le nom du serveur et surtout deux regexp qui vont nous transformer le nom d'utilisateur au « format » SASL en dn compréhensible par notre serveur LDAP. Il faut le faire en deux fois car l'utilisateur **admin** n'est pas dans l'ou **people** mais directement sous **dc=test,dc=glmf**. Cyrus SASL présente systématiquement le nom de connexion sous la forme **uid=<user>@<realm>,cn=<realm>,cn=<mode d'authentification>,cn=auth** et il faut transformer ça en **cn=<user>,ou=people,dc=test,dc=glmf**. C'est l'objet de la deuxième regexp dans laquelle **$1** représente la partie sauvegardée (entre parenthèse), c'est à dire le nom de l'utilisateur.

Une fois la configuration recréé grâce à **slaptest**, nous pouvons redémarrer le serveur **slapd** et effectuer notre premier test. Ici nous sommes donc en mode chiffré TLS ET authentification SASL. Il faut donc changer encore un peu nos options pour **ldapsearch** et cela donne :

```
# ldapsearch -ZZ -U admin@openldap -b 'dc=test,dc=glmf' '(objectclass=*)'
```

C'est ici l'option **-U** qui indique qu'il faut passer par SASL et le **@** sépare le nom de base de l'utilisateur (et non plus le **dn** complet) du nom du royaume.

Normalement tout devrait bien se passer. Si jamais un problème se présentait, n'oubliez pas que vous pouvez lancer **slapd** à la main en rajoutant l'option **-d 3**[12] pour le debug.

Pour optimiser votre serveur LDAP vous pouvez maintenant vous attaquer aux indexes. Comme nous l'avons déjà entre-aperçu plus haut, cela se passe dans **slapd.conf** ou dans **slapd.d/cn=config/olcDatabase={1}bdb.ldif**. Les objets à indexer sont fonction de vos futures requêtes et des schémas que vous utilisez. La documentation est facilement trouvable sur le net mais dépasse le cadre de cet article. ■

[11] http://en.wikipedia.org/wiki/Simple_Authentication_and_Security_Layer

[12] La valeur peut varier et est en fait un masque binaire entre 0 et 0xFFFF mais 3 est bien pour ce qui nous intéresse.

ON A PERDU LE MINITEL MAIS ON A TOUJOURS L'ANNUAIRE...

LDAP : 2 - LE CÔTÉ CLIENT

par Cédric PELLERIN [Utilisateur de GNU/Linux depuis 1993]

Ou comment « ldapifier » certaines de nos applications favorites. Le problème que l'on rencontre lorsque l'on veut connecter une application cliente à OpenLDAP est que toutes les options peuvent ne pas être supportées. Un exemple récent est le logiciel Keystone qui est le système d'authentification centralisé d'OpenStack. Il est parfaitement connectable à un annuaire tant que la connexion n'est pas chiffrée par TLS.

La demande de correction qui a été faite en août 2012 n'a été implémenté qu'en mai 2013 mais l'absence chronique de documentation fiable sur ce projet m'a empêché de valider son fonctionnement pour cet article.

Cependant, beaucoup de logiciels majeurs et qui fonctionnent bien sont connectables sans souci.

1 Les utilitaires

Commençons par expliquer un peu plus en détail certains utilitaires livrés avec OpenLDAP. Vous remarquerez qu'on peut les séparer en deux groupes, ceux stockés dans **sbin** et qui commencent par « slap » et ceux stockés dans **bin** qui commencent par « ldap ». Les premiers, dont les plus utilisés sont **slapadd** et **slapcat**, attaquent directement la base de donnée sans passer par le serveur OpenLDAP. Cela signifie qu'ils doivent être utilisés sur le serveur même qui fait tourner slapd, que l'utilisateur doit avoir les droits sur la base et que l'on contourne toutes les procédures d'authentification d'OpenLDAP. Un effet secondaire important est que, si l'on modifie la base de données (via **slapadd** par exemple) il est impératif que **slapd** soit arrêté. Ceci est moins fondamental quand on se contente de la consulter via **slapcat**. Vous comprenez pourquoi ils sont dans **sbin**, répertoire qui n'est, en général, accessible qu'à **root**.

En revanche les seconds, parmi lesquels on utilise surtout **ldapadd**, **ldapmodify**, **ldappasswd** et **ldapsearch**, utilisent le protocole LDAP pour se connecter au serveur. Ceci veut dire que l'on peut se connecter à un serveur distant et que l'on devra passer au travers des mécanismes d'authentification LDAP.

Ces utilitaires sont dotés d'une multitude d'options car il faut s'adapter à toutes les configurations OpenLDAP possibles et il y en a un paquet. Examinons rapidement les plus utiles outil par outil.

Tout d'abord nous avons **slapadd** :

- **-b <suffix>** précise à quel suffixe il faut ajouter les données.
- **-c** continue même s'il rencontre des erreurs en cours de route.
- **-f** indique quel fichier de configuration utiliser. Il s'agit de **slapd.conf** par défaut.
- **-F** comme **-f** mais dans le cas d'un répertoire pour une configuration de type **cn=config.**
- **-l** précise le fichier LDIF à insérer.

Exemple :

```
# slapadd -b dc=test,dc=glmf -l user25.ldif
```

insère le contenu du fichier **user25.ldif** sous l'arborescence **dc=test,dc=glmf**

Examinons maintenant **slapcat** :

Les options **-c**, **-f** et **-F** sont les mêmes que pour **slapadd**. Les options **-b** et **-l** sont semblables mais leur résultat n'est pas le même :

- **-b <suffix>** précise la base que l'on veut afficher
- **-a <filter>** rajoute un filtre pour la sortie
- **-l** précise le fichier LDIF dans lequel **slapcat** écrira les données au lieu de la sortie standard

Exemple :

```
# slapcat -b dc=test,dc=glmf
```

L'outil de recherche est **ldapsearch** :

- **-L** Par défaut, les résultats sont imprimés au format LDIF étendu avec tous les en-têtes. **-L** restreint la sortie au format LDIF v1, **-LL** supprime les commentaires, -LLL supprime l'impression de la version de LDIF. L'usage de l'option **-LLL** permet de n'avoir que les résultats

sans tout un tas de commentaires inutiles dans lesquels on risque de se perdre.

- **-b <base-dn>** spécifie la base de recherche
- **-D <binddn>** précise le DN de l'utilisateur avec lequel effectuer la recherche. Celui-ci doit avoir les droits sur l'arborescence dans laquelle on veut travailler.
- **-x** utilise l'authentification simple (**password**) au lieu de SASL par défaut, en collaboration avec **-w <password>** qui permet de préciser le mot de passe de connexion sur la ligne de commande ou **-W** qui demandera de le saisir
- **-H <ldapuri>** permet la connexion au serveur via une URI de type **ldap://<serveur:port>/**. Il s'agit de la version « moderne » à préférer aux dépréciées **-h <serveur>** et **-p <port>**
- **-U <username[@realm]>** permet de préciser le nom d'utilisateur pour une connexion SASL. Le realm peut aussi être spécifié séparément via l'option **-R <realm>**.
- **-Z** Démarre une connexion sécurisée de type TLS. **-ZZ** exige que l'opération soit complètement valide (c'est à dire que **-ZZ** échouera par exemple si le FQDN du serveur diffère de celui inscrit dans le certificat utilisé).

L'outil **ldapsearch** prend ensuite un filtre plus ou moins complexe qui permet de trier les résultats voulus et la liste des attributes à afficher. Exemple :

```
# ldapsearch -U admin@openldap -ZZ -b
ou=people,dc=test,dc=glmf "cn=toto" gecos uid
```

permettra d'afficher les attributes **gecos** et **uid** de l'utilisateur **toto** situé dans l'arborescence **ou=people,dc=test,dc=glmf**.

Les filtres peuvent être plutôt complexes et leur syntaxe est tout sauf intuitive. Vous pourrez trouver de plus amples informations ici http://www.ldapguru.info/ldap/mastering-ldap-search-filters.html

Et enfin, nous avons **ldapmodify**[1] :

Les options de connexion (**-D**, **-x**, **-w**, **-W**, **-Z**, **-U**, **-H**, **-h**, **-p**) sont les mêmes que pour **ldapsearch**. Cependant **ldapmodify** prend aussi d'autres options utiles :

- **-f <file>** spécifie le fichier LDIF à utiliser pour les modifications. Sa syntaxe est un peu spéciale et est décrite ci-dessous.
- **-c** permet de continuer en cas d'erreurs. Par défaut **ldapmodify** s'arrête à la première erreur rencontrée.
- **-S <file>** permet de stocker les entrées en erreur. Utile avec l'option **-c** ci-dessus.

Comme indiqué ci-dessus, le fichier LDIF à utiliser avec **ldapmodify** possède une syntaxe un peu différente de celle déjà rencontrée. Un exemple sera plus parlant qu'une austère description grammaticale :

```
dn: cn=toto,ou=people,dc=test,dc=glmf
changetype: modify
replace: mail
mail: toto2@test.glmf
-
add: title
title: Utilisateur de demo
-
add: jpegPhoto
jpegPhoto:< file:///tmp/toto.jpeg
-
delete: description
-
```

Une fois ce fichier envoyé à **ldapmodify** l'utilisateur **toto** verra son mail modifié, son titre et sa photo ajoutée et sa description supprimée.

Il existe d'autres utilitaires en ligne de commande livrés avec les sources d'OpenLDAP ou disponibles via le paquet **ldap-utils** que je vous laisse le soin de découvrir.

Pour leurs options, tous ces utilitaires utilisent le fichier **ldap.conf** situé dans **/etc/ldap** ou **/usr/local/etc/openldap** qui concentre les paramètres liés aux clients comme, par exemple, le certificat à utiliser pour les connexions SSL et TLS. Ce fichier est censé être utilisé par tous les clients LDAP mais c'est plus ou moins vrai. A vérifier au cas par cas.

Pour les réfractaires à la ligne de commande il existe de nombreuses interfaces graphiques plus ou moins finies et plus ou moins utilisables. Une recherche sur le grand ternet devrait vous ramener un bon paquet de liens. A vous de faire le tri et de voir si vous trouvez votre bonheur. Cependant ne vous faites pas trop d'illusions, la liberté d'implémentation offerte par LDAP est telle qu'il vous faudra sans doute adapter vous-même la GUI à vos besoins, ou alors accepter de passer parfois par la case « terminal ».

2 Cyrus SASL

Nous avons déjà utilisé Cyrus SASL comme système d'authentification SASL pour notre serveur LDAP. Cependant cette suite est nettement plus puissante que cela et peut servir de centralisation d'authentification pour les produits la supportant. Nous allons donc, à titre d'exemple, mettre en place un serveur SASL Cyrus qui se servira d'OpenLDAP comme backend pour la gestion des comptes. Nul n'est besoin de rajouter des paquets par rapport à la section précédente, il suffit de configurer le serveur SASL nommé **saslauthd** et l'activer.

Nous partons du principe que nous allons nous connecter au serveur OpenLDAP via TLS et authentification SASL pour la recherche[2]. La configuration se fait dans le fichier **/etc/saslauthd.conf** qu'il faut créer. Ce fichier doit contenir les lignes suivantes :

[1] ldapadd n'est qu'un appel à ldapmodify avec l'option -a pour effectuer uniquement des ajouts.

[2] Oui je sais, un serveur SASL qui se connecte en SASL pour faire sa recherche ça fait bizarre, et pourtant...

```
ldap_servers: ldap://tyr.test.glmf
ldap_search_base: dc=test,dc=glmf
ldap_version: 3
ldap_auth_method: fastbind
ldap_filter: cn=%u,ou=people,dc=test,dc=glmf
ldap_bind_dn: cn=admin,dc=test,dc=glmf
ldap_bind_pw: password
ldap_start_tls: yes
ldap_tls_check_peer: yes
ldap_tls_cert: /etc/ssl/certs/tyr.pem
ldap_tls_cacert_dir: /etc/ssl/certs
ldap_tls_key: /usr/local/etc/openldap/private.pem
ldap_use_sasl: yes
```

Ce contenu se passe assez facilement de commentaires. On indique l'adresse du serveur LDAP à qui on ira demander les comptes et mots de passe, la base de recherche (le **-b** de **ldapsearch**), le filtre pour récupérer les noms des utilisateurs, le nom et le mot de passe avec lesquels se connecter pour effectuer la recherche, si on est en TLS ou en SSL (**start_tls**), là où trouver les clés et certificats et enfin que l'on se connecte en SASL.

Ce fichier créé, il faut aller effectuer deux modifications dans le fichier **/etc/default/saslauthd** :

```
START=yes
MECHANISMS="ldap"
```

et ajouter **-O /etc/saslauthd.conf** à la fin de la directive **OPTIONS= à la fin du même fichier**.

Cela fait on relance le serveur SASL :

```
# /etc/init.d/saslauthd restart
```

et on fait le test qui va bien :

```
# testsaslauthd -u cpellerin -p password
```

ce qui devrait nous répondre :

```
0: OK "Success."
```

N'essayez pas avec le user **admin** car il n'est pas sous l'OU **people** et le filtre ne fonctionnera donc pas.

Si cela ne se passe pas comme prévu, relisez bien le fichier **saslauthd.conf** car une faute de frappe est vite venue et sournoise. Vous pouvez aussi relancer OpenLDAP en mode debug dans un terminal, **saslauthd** aussi en mode debug dans un autre via la commande :

```
# saslauthd -a ldap -O /etc/saslauthd.conf -d
```

et refaire vos tests dans un troisième. En analysant calmement les sorties de nos deux daemons, vous devriez trouver l'erreur. Attention à la syntaxe du fichier **saslauthd.conf**, le séparateur clef/valeur est le **:** pas le **=**.

Pour ceux qui ne peuvent pas utiliser TLS, il faut juste effectuer les modifications suivantes dans le fichier **/etc/saslauthd.conf** pour utiliser la connexion SSL :

```
ldap_servers: ldaps://tyr.test.glmf
ldap_start_tls: no
ldap_tls_check_peer: no
```

3 Postfix

La manière la plus simple pour que Postfix puisse faire de l'authentification de manière sécurisée est d'utiliser SASL. Il est souvent très lié à Dovecot-SASL mais comme nous avons déjà installé Cyrus-SASL nous allons utiliser ce dernier.

3.1 Pré-requis

Commençons par installer Postfix via un bon vieux :

```
# apt-get install postfix
```

A la question de l'installeur concernant le type de serveur, répondons « internet » afin d'avoir déjà un début de configuration opérationnel puis allons modifier le fichier principal nommé **/etc/postfix/main.cf**. Le but est de servir de serveur SMTP final et de n'envoyer le courrier que pour notre réseau à nous. Ce dernier a été repris du précédent article et est donc toujours 172.21.0.0/22. Ces considérations nous donnent le **main.cf** suivant :

```
myorigin = /etc/mailname
smtpd_banner = You talkin' to me ?
biff = no
append_dot_mydomain = no
readme_directory = no

myhostname = tyr.test.glmf
alias_maps = hash:/etc/aliases
alias_database = hash:/etc/aliases
mydestination = tyr.test.glmf, test.glmf, localhost
relayhost =
mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128
172.21.0.0/22
mailbox_command = procmail -f- -a $USER
mailbox_size_limit = 0
recipient_delimiter = +
inet_interfaces = all
inet_protocols = ipv4

header_checks = regexp:/etc/postfix/mime_header_checks.regexp
#smtpd_helo_restrictions = reject_unknown_helo_hostname
smtpd_sender_restrictions = reject_unknown_sender_domain, reject_non_
fqdn_sender
#smtpd_client_restrictions = reject_unauth_pipelining
smtpd_recipient_restrictions = reject_unknown_recipient_domain,
reject_non_fqdn_recipient, permit_mynetworks, permit_sasl_
authenticated, reject_unauth_destination, permit
# Ou pour Postfix > 2.9
#smtpd_relay_restrictions = reject_unknown_recipient_domain, reject_
non_fqdn_recipient, permit_mynetworks, permit_sasl_authenticated,
reject_unauth_destination, permit
smtpd_helo_required = yes
message_size_limit = 20480000
```

Avec ce fichier associé à un fichier **/etc/postfix/aliases** de base du genre :

```
mailer-daemon: postmaster
postmaster: root
nobody: root
hostmaster: root
usenet: root
news: root
webmaster: root
www: root
ftp: root
abuse: root
noc: root
security: root
root: admin@test.glmf
```

compilé par **postalias** et un fichier **/etc/procmailrc** simpliste du genre :

```
LOGNAME=$USER
SHELL=/bin/sh
PATH=/bin:/usr/bin:/usr/local/bin:$HOME/bin

MAILDIR=$HOME/Maildir
DEFAULT=$HOME/Maildir/cur
ORGMAIL=$MAILDIR/emergency-inbox
LOGFILE=/var/log/procmail.log
DROPPRIVS=yes

VERBOSE=no

:0
* ^From: admin@test.glmf
| $DELIVER
```

nous avons un serveur Postfix fonctionnel[3] mais qui ne requiert aucune authentification, ce qui peut être dangereux s'il est ouvert sur l'extérieur. Il nous faut donc verrouiller un peu tout ça et on va en profiter pour utiliser notre LDAP tout neuf.

3.2 Authentification via saslauthd

Afin d'utiliser notre serveur Cyrus-SASL, nous devons expliquer quelques petites choses à Postfix, à rajouter à la fin du fichier **main.cf** :

```
smtpd_sasl_auth_enable = yes
smtpd_sasl_type = cyrus
smtpd_sasl_security_options = noanonymous
smtpd_sasl_path = smtpd
smtpd_sasl_local_domain = openldap
```

Rien de bien compliqué, ça se passe de commentaire. La directive **smtpd_sasl_local_domain**, comme son nom ne l'indique pas, désigne en fait le royaume SASL et non le domaine DNS.

Ceci fait, les choses se corsent un petit peu. En effet il faut aller créer le fichier **/etc/postfix/sasl/smtpd.conf**[4] et y mettre juste la ligne suivante :

```
pwcheck_method: saslauthd
```

Ensuite nous sommes confrontés à un problème de droits, du moins sous Debian, lié au **chroot** fait par Postfix. Pour que tout fonctionne, il faut donc faire le petit hack goret suivant :

```
# rm -r /var/run/saslauthd/
# mkdir -p /var/spool/postfix/var/run/saslauthd
# ln -s /var/spool/postfix/var/run/saslauthd /var/run
# chgrp sasl /var/spool/postfix/var/run/saslauthd
# adduser postfix sasl
```

ce qui va permettre à Postfix de communiquer avec **saslauthd**.

Une fois tout ceci effectué, nous pouvons relancer Postfix et SASLauthd et aller tester via telnet que l'authentification fonctionne. Pour ce faire nous aurons besoin de l'encodage en base 64 de la chaîne d'authentification attendue par saslauthd. Elle doit être de la forme **encode_base64("username\0username\0password")**. Pour la générer on va demander gentiment à Perl. Par exemple avec l'utilisateur **cpellerin** :

```
$ perl -MMIME::Base64 -e 'print encode_base64("cpellerin\0cpellerin\
0password");'
```

ce qui nous donne comme réponse :

```
Y3BlbGxlcmluAGNwZWxsZXJpbgBwYXNzd29yZA==
```

Maintenant que nous avons tous les éléments, on teste :

```
$ telnet localhost 25
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 You talkin' to me ?
ehlo tyr.test.glmf
250-tyr.test.glmf
250-PIPELINING
250-SIZE 20480000
250-VRFY
250-ETRN
250-AUTH DIGEST-MD5 CRAM-MD5 NTLM LOGIN PLAIN
250-AUTH=DIGEST-MD5 CRAM-MD5 NTLM LOGIN PLAIN
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
auth plain Y3BlbGxlcmluAGNwZWxsZXJpbgBwYXNzd29yZA==
235 2.7.0 Authentication successful
quit
221 2.0.0 Bye
```

et voilà, le tour est joué. Nous avons donc pu valider toute la chaîne depuis le client mail émulé ici en **telnet** jusqu'à l'authentification par **saslauthd** avec un compte et un mot de passe stockés dans notre annuaire LDAP.

4 Cyrus-imap

Comme nous avons commencé avec Cyrus-SASL, autant continuer avec le même « fournisseur » pour la partie IMAP de notre serveur. La version disponible sous Debian est la 2.4 qui s'installe de la manière suivante :

[3] Le but ici n'est pas de détailler la configuration de Postfix qui a déjà été vue dans un article précédent. Les fichiers sont donc donnés là à titre purement indicatifs afin que vous puissiez monter une configuration fonctionnelle de test.

[4] Le nom du fichier est important. C'est celui donné via la directive smtpd_sasl_path auquel on rajoute l'extension .conf et il doit être situé dans /etc/postfix/sasl pour une distribution Debian-like.

```
# apt-get install cyrus-admin cyrus-clients cyrus-imapd
```

Avant de configurer le serveur IMAP, nous allons rajouter un utilisateur dans notre annuaire, utilisateur qui sera destiné à administrer le serveur. Pour ce faire, il nous faut créer le fichier **cyradmin.ldif** suivant :

```
dn: cn=cyrus,ou=people,dc=test,dc=glmf
objectClass: top
objectClass: posixAccount
objectClass: shadowAccount
objectClass: inetOrgPerson
uid: cyrus
sn: Cyrus
uidNumber: 1002
gidNumber: 1002
userPassword: password
gecos: Cyrus admin
loginShell: /bin/bash
homeDirectory: /home/cyrus
mail: cyrus@test.glmf
```

Si nous comparons avec notre précédente insertion d'utilisateur, nous avons rajouté le support du schéma **inetOrgPerson** afin de disposer de l'attribute **mail:**. Le support de ce schéma nous oblige à rajouter l'attribute **sn:** qui prend simplement le nom courant de l'utilisateur. Cette nouvelle entry se rajoute classiquement par la commande :

```
# ldapadd -ZZ -U admin@openldap -f cyradmin.ldif
```

Une fois ceci effectué, nous pouvons aller éditer le fichier **/etc/imapd.conf** et modifier ou rajouter les lignes suivantes :

```
admins: cyrus
allowplaintext: yes
sasl_mech_list: PLAIN
sasl_pwcheck_method: saslauthd
tls_cert_file: /etc/ssl/certs/OpenLDAP.pem
tls_key_file: /etc/ssl/private/private.pem
imap_tls_cert_file: /etc/ssl/certs/OpenLDAP.pem
imap_tls_key_file: /etc/ssl/private/private.pem
tls_ca_path: /etc/ssl/certs
```

On retrouve ici l'utilisateur cyrus que l'on vient de rajouter ainsi que la méthode d'authentification SASL et les clés et certificats nécessaires.

Avant de relancer Cyrus-IMAP, il nous faut effectuer quelques réglages. Tout d'abord créer le répertoire **/run/cyrus/lock** et l'affecter à l'utilisateur **cyrus** :

```
# mkdir /run/cyrus/lock
# chown -R cyrus /run/cyrus
```

ensuite il faut mettre **cyrus** dans le groupe **ssl-cert** afin que le serveur aie le droit de lire la clé privée :

```
# adduser cyrus ssl-cert
```

et, pour finir, il faut aller modifier la configuration du serveur ldap afin de rajouter un filtre car cyrus-imap ne rajoute pas le **@<realm>** derrière le nom d'utilisateur lors de la connexion ce qui fait planter l'authentification. Pour effectuer cette démarche il suffit de modifier le fichier **slapd.conf** en rajoutant la ligne :

```
authz-regexp uid=(.*),cn=openldap,cn=digest-md5,cn=auth
cn=$1,ou=people,dc=test,dc=glmf
```

puis de recréer la configuration avec **slaptest**.

Il est maintenant temps de relancer slapd et cyrus-imap et de vérifier que tout fonctionne. Le premier test va se faire avec l'utilitaire **testsaslpasswd** pour valider nos filtres :

```
# testsaslauthd -u cpellerin -r openldap -p password
0: OK "Success."
# testsaslauthd -u cpellerin -p password
0: OK "Success."
```

Parfait, avec ou sans présentation de realm, l'authentification se fait bien.

Essayons maintenant côté cyrus-imap et, si ça marche, profitons en pour créer notre utilisateur **cpellerin** côté imap. Cela se fait ainsi :

```
# cyradm --user cyrus localhost
Password: password
localhost> cm user.cpellerin
localhost> lm
user.cpellerin (\HasNoChildren)
localhost> quit
```

Si vous avez un I/O error sur la commande **cm**, vérifiez bien que **/run/cyrus/lock** est un REPERTOIRE et qu'il appartient bien à **cyrus**. Pour le debogage, les logs de cyrus sont dans **syslog** par défaut.

Maintenant essayons de nous connecter au serveur IMAP via **telnet** :

```
# telnet localhost 143
Trying ::1...
Connected to localhost.
Escape character is '^]'.
* OK [CAPABILITY IMAP4rev1 LITERAL+ ID ENABLE STARTTLS AUTH=PLAIN SASL-IR]
OpenLDAP Cyrus IMAP v2.4.16-Debian-2.4.16-4+deb7u1 server ready
01 login cpellerin password
01 OK [CAPABILITY IMAP4rev1 LITERAL+ ID ENABLE ACL RIGHTS=kxte QUOTA
MAILBOX-REFERRALS NAMESPACE UIDPLUS NO_ATOMIC_RENAME UNSELECT CHILDREN
MULTIAPPEND BINARY CATENATE CONDSTORE ESEARCH SORT SORT=MODSEQ
SORT=DISPLAY THREAD=ORDEREDSUBJECT THREAD=REFERENCES ANNOTATEMORE LIST-
EXTENDED WITHIN QRESYNC SCAN XLIST URLAUTH URLAUTH=BINARY LOGINDISABLED
COMPRESS=DEFLATE IDLE] User logged in SESSIONID=<cyrus-15161-1371818924-1>
02 list "" *
* LIST (\HasNoChildren) "." INBOX
02 OK Completed (0.000 secs 2 calls)
03 logout
* BYE LOGOUT received
03 OK Completed
Connection closed by foreign host.
```

Youpi, tout roule. Si vous avez le courage vous pouvez valider avec Thunderbird, Kmail ou autre Mutt que la connexion s'effectue bien.

Notre serveur SMTP/IMAP est donc fonctionnel en ce qui concerne les authentifications. Pour le rendre fonctionnel à 100 % il faut maintenant le déclarer en tant que MX dans votre DNS, ouvrir le port 25 sur votre firewall et sans doute retoucher quelques paramètres, principalement en ce qui concerne la distribution du courrier. Ce sera donc notre exercice de fin de chapitre, vous avez quatre heures !

5 Samba

Pour finir, nous allons étudier rapidement les modifications à apporter à un serveur Samba afin qu'il puisse se servir d'un annuaire LDAP. Si votre serveur n'est pas encore installé, un classique :

```
# apt-get install samba smb-client
```

suffira. Dans tous les cas, il nous faut aussi installer le paquet **samba-doc** afin d'y récupérer le schéma LDAP pour samba :

```
# apt-get install samba-doc
```

Ceci fait, nous pouvons copier le fichier dans notre répertoire et le décompresser :

```
# cp /usr/share/doc/samba-doc/examples/LDAP/
samba.ldif.gz ~
# gunzip samba.ldif.gz
```

puis nous l'ajoutons dynamiquement au serveur, sans avoir besoin de le redémarrer :

```
# ldapadd -ZZ -x -w password -D
'cn=manager,cn=config' -f samba.ldif
```

C'est beau le progrès !

Pour fonctionner, Samba a besoin d'une OU supplémentaire pour stocker les comptes des ordinateurs, OU que nous allons appeler **Computers**. Donc, on crée le ldif qui va bien :

```
# vi smbou.ldif
```

```
dn: ou=Computers,dc=test,dc=glmf
objectClass: organizationalUnit
ou: Computers
```

et on l'injecte :

```
# ldapadd -ZZ -U admin@openldap -f smbou.ldif
adding new entry "ou=Computers,dc=test,dc=glmf"
```

Les modifications à apporter au fichier **/etc/samba/smb.conf** se résument à deux choses :

1. Modifier la directive **passdb backend** pour lui dire d'utiliser LDAP :

```
passdb backend = ldapsam:ldap://tyr.test.glmf:389
```

2. Rajouter les lignes suivantes concernant LDAP :

```
ldap admin dn = "cn=admin,dc=test,dc=glmf"
ldap suffix = dc=test,dc=glmf
ldap delete dn = no
unix password sync = yes
ldap user suffix = ou=people
ldap group suffix = ou=groups
ldap machine suffix = ou=Computers
ldap ssl = start tls
```

Il est à noter que si l'**admin dn** peut s'écrire entre guillemets sans problème il ne faut surtout pas le faire pour les suffixes sous peine de les voir inclus dans le DN final et donc poser quelques soucis...

Après ces travaux herculéens, il faut relancer samba puis lui fournir le mot de passe pour l'**admin dn**. Vous n'aurez pas été sans remarquer qu'il n'est pas stocké dans le fichier **smb.conf**, c'est plus prudent. Samba va le déposer soigneusement dans un fichier nommé **secrets.tdb** grâce à la commande suivante :

```
# smbpasswd -w password
```

Ensuite nous allons créer un utilisateur nommé **titi**, d'un côté sous Linux :

```
# adduser titi
```

et de l'autre sous Samba :

```
# smbpasswd -D10 -a titi
```

Le **-D10** est là pour pouvoir suivre la conversation avec OpenLDAP et réagir si quelque chose se déroule mal.

Pour finir, un petit test simple :

```
# smbclient -U titi //tyr.test.glmf/titi
Enter titi's password:
Domain=[GLMF] OS=[Unix] Server=[Samba 3.6.15]
smb: \>
```

et hop tout fonctionne.

Pour ceux qui n'ont pas le support TLS mais qui ont mis en place le SSL via ldaps, il faut modifier les deux lignes suivantes :

```
passdb backend = ldapsam:ldaps://tyr.test.glmf:636
ldap ssl = off
```

Conclusion

Nous avons pu voir au cours de cet article et du précédent comment monter un serveur OpenLDAP, l'alimenter en données et configurer certains clients. Il y aurait encore de très nombreuses choses à dire mais j'entends déjà notre rédac'chef préféré râler qu'il ne va pas savoir comment caler un monstre pareil dans GLMF ;). Pour continuer votre exploration, une piste très intéressante s'appelle **pam_ldap**. Ce module permet de rajouter le support LDAP à PAM et ainsi vous permettre de vous connecter sous Linux avec authentification directe sur OpenLDAP. Ce module, lié à **automount**, permet à n'importe qui de se connecter sur n'importe quelle machine tout en retrouvant ses petits sans problème. C'est parfois assez complexe mais très amusant voire utile.

Vous pouvez aussi aller voir comment Bind sait stocker ses informations dans un annuaire ou essayer de faire fonctionner Dovecot (j'ai renoncé à cause du SASL qui ne fonctionne pas avec Cyrus-SASL mais sans SASL ça se fait bien). Bref avec un peu de ténacité, OpenLDAP peut très vite se retrouver au cœur de votre réseau et il sera alors temps de regarder comment ça se sauvegarde et comment on crée un serveur esclave en cas de crash du principal. Un conseil, gardez quand même un accès root local sur vos machines, on ne sait jamais... ■

LE COIN DU VIEUX BARBU :

UNE CALCULATRICE PLEINE DE SURPRISES !

par David Odin [Si barbu qu'il a toujours sa bonne vieille HP28S avec un port série en morse fait main]

Le programme du jour permet de s'apercevoir que ce qui paraît simple au premier abord peut en fait regorger d'astuces aussi tordues les unes que les autres. Quoi de plus simple qu'une calculatrice ? Celle que Hou Qiming a proposé à l'IOCCC en 2011 semble bien innocente et pourtant...

Je voulais me remettre de ma déconvenue de la dernière fois avec les constantes incompréhensibles du simulateur de vol de M. Banks. Je me tourne aujourd'hui vers quelque chose qui me semble plus simple, et dont les constantes sont bien connues : une calculatrice multifonction supportant les quatre opérations de base (avec respect des précédences), les parenthèses et quelques fonctions mathématiques classiques. Cette entrée a gagné le concours de l'IOCCC **[1]** en 2011 dans la catégorie « *Best self documenting program* ».

1 Version initiale

Voici la version proposée à l'IOCCC **[1]**, sans changement, que vous pouvez retrouver sur http://forma3dev.fr/vieux-barbu/ :

```
#include <stdio.h>
#include <math.h>
#define clear 1;if(c>=11)
{c=0;sscanf(_,"%lf%c",&r,&c);while(*++_-c);}\
  else if(argc>=4&&!main(4-(*_++=='('),argv))_++;g:c+=
#define puts(d,e) return 0;}{double a;int b;char c=(argc<4?d)&15;\
  b=(*_%__LINE__+7)%9*(3*e>>c&1);c+=
#define I(d) (r);if(argc<4&&*#d==*_){a=r;r=usage?r*a:r+a;goto g;}c=c
#define return if(argc==2)printf("%f\n",r);return argc>=4+
#define usage main(4-__LINE__/26,argv)
#define calculator *_*(int)
#define l (r);r=--b?r:
#define _ argv[1]
#define x

double r;
int main(int argc,char** argv){
  if(argc<2){
    puts(
      usage: calculator 11/26+222/31
      +~~~~~~~~~~~~~~~~~~~~~~~~~~calculator-\
      !                          7.584,367 )
      +~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+
      ! clear ! 0 ||1   -x  1   tan  I (/) |
      +~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+
      ! 1 | 2 | 3 ||1  1/x  1   cos  I (*) |
      +~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+
      ! 4 | 5 | 6 ||1  exp  1  sqrt  I (+) |
      +~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+
      ! 7 | 8 | 9 ||1  sin  1   log  I (-) |
      +~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~(0
    );
  }
  return 0;
}
```

Au premier abord, on peut avoir l'impression que ce programme ne fait qu'afficher une calculatrice en ASCII-Art si on ne lui passe pas d'argument, et rien d'autre dans le cas contraire. Mais alors, où serait le code ?

Pourtant, si on compile cela (avec un **`gcc hou.c -lm`**) et que l'on exécute le programme avec un calcul en argument, on obtient bien un résultat :

```
$ ./a.out '1+1'
2.000000
$ ./a.out 'sin(3.141592/2)'
1.000000
$ ./a.out '1+1/2-3*9'
-25.500000
$ ./a.out 'exp(5.39231742*log(2)'
42.000000
```

En y regardant de plus près, ce qui ressemble au paramètre de la fonction **`puts`** n'est pas du tout une chaîne de caractères ! Il s'agit en fait de l'ensemble du code. **`puts`** est en fait une macro, dont le début (**`usage`**) est également une macro.

On a donc fait un pas de plus dans l'auto-documentation, puisque le code source est en fait la documentation utilisateur !

2 Premier champ de mines : des #define avec des __LINE__

Pour arriver à comprendre ce programme, il va donc falloir commencer par remplacer toutes les macros (les **#define**) par leur contenu. Mais l'auteur a prévu pas mal de pièges pour éviter de nous rendre les choses trop faciles.

En effet, l'auteur utilise fréquemment les symboles **l**, **I**, **1**, **!** et **|** ce qui n'est pas bien grave, mais rend le tout difficile à lire si l'on a pas une police de caractères très lisible.

Certaines macros en appellent d'autres. Par exemple, la macro **calculator** utilise la macro **_**. Encore une fois, ce n'est pas méchant, mais il va falloir faire attention à l'ordre de remplacement des macros par leur valeur.

La macro **return** a comme nom un mot réservé du C (ce qui est permis par le langage mais évidemment pas recommandé !). Cela signifie que ce mot-clef ne pourra pas être utilisé, sauf éventuellement dans le corps de la macro comme c'est le cas ici.

En C, une macro a des paramètres si, lors de la définition, son nom est immédiatement suivi par une parenthèse ouvrante. Ainsi, la macro **I(d)** prend un seul paramètre ; elle est utilisée en prenant comme paramètre chacune des opérations de base. Notons que le paramètre **d** n'est utilisé dans le corps de la macro que précédé d'un dièse. Cette notation indique qu'il faudra remplacer le **#d** par une chaîne de caractères contenant le paramètre. Par exemple, dans l'extension de **I (/)** (notez que l'espace avant la parenthèse est ici sans effet), la partie ***#d==*_** devra être remplacé par ***"/"==*_**.

La macro **puts(d,e)** prend deux paramètres, **d** et **e**. Cette macro n'est utilisée qu'une seule fois, mais ses paramètres sont peu communs. Les paramètres sont forcément séparés par des virgules, donc le premier commence par **usage** (encore une macro !) et se termine par **7.584**. Le second est simplement **367**.

La macro **l** ressemble à une macro avec paramètre, ce n'est pourtant pas le cas car il y a un espace entre le **l** et la parenthèse. Elle doit donc être simplement remplacée par **(r);r=--b?r:** dans les 8 cas où elle est rencontrée.

Enfin, certaines macros (**puts** et **usage**) utilisent la valeur spéciale **__LINE__**. Cette valeur contient le numéro de la ligne du fichier dans laquelle la macro apparaît après l'extension des macros. Et comme **usage** est appelée par **I**, il faut faire attention à ne pas ajouter ni enlever de ligne avant le remplacement de toutes les macros.

En prenant toutes ces précautions, on arrive à la version suivante :

```
001  #include <stdio.h>
002  #include <math.h>
003  //#define clear 1;if(c>=11){c=0;sscanf(_,"%lf%c",&r,&c);
                                                  while(*++_-c);} \
```

```
004  //  else if(argc>=4&&!main(4-(*_++=='('),argv))_++;g:c+=
005  //#define puts(d,e) return 0;}{double a;int b;char c=(argc<4?d)
                                                              &15;\
006  //  b=(*_%__LINE__+7)%9*(3*e>>c&1);c+=
007  //#define I(d) (r);if(argc<4&&*#d==*_){a=r;r=usage?r*a:r+a;goto
                                                            g;}c=c
008  //#define return if(argc==2)printf("%f\n",r);return argc>=4+
009  //#define usage main(4-__LINE__/26,argv)
010  //#define calculator *_*(int)
011  //#define l (r);r=--b?r:
012  //#define _ argv[1]
013  //#define x
014
015  double r;
016  int main(int argc,char** argv){
017    if(argc<2){
018      if(argc==2)printf("%f\n",r);return argc>=4+0;}{double a;int b;
             char c=(argc<4?main(4-__LINE__/26,argv): *argv[1]*(int)
                                                          11/26+222/31
019        +~~~~~~~~~~~~~~~~~~~~~~~~*argv[1]*(int)-\
020        !                        7.584
021          )&15; b=(*argv[1]%__LINE__+7)%9*(3*367>>c&1);c+=
022        +~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+
023        ! 1;if(c>=11){c=0;sscanf(argv[1],"%lf%c",&r,&c);
           while(*++argv[1]-c);} else if(argc>=4&&!main(4-(*argv[1]++==
           '('),argv))argv[1]++;g:c+= ! 0 ||(r);r=--b?r:   -  (r);r=--b
           ?r:   tan  (r);if(argc<4&&*"/"==*argv[1]){a=r;r=main(4-
           __LINE__/26,argv)?r*a:r+a;goto g;}c=c|
024        +~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+
025        ! 1 | 2 | 3 ||(r);r=--b?r:  1/  (r);r=--b?r:   cos  (r);
           if(argc<4&&*"*"==*argv[1]){a=r;r=main(4-__LINE__/26,argv)?
           r*a:r+a;goto g;}c=c|
026        +~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+
027        ! 4 | 5 | 6 ||(r);r=--b?r:  exp  (r);r=--b?r:  sqrt (r);
           if(argc<4&&*"+"==*argv[1]){a=r;r=main(4-__LINE__/26,argv)?
           r*a:r+a;goto g;}c=c|
028        +~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+
029        ! 7 | 8 | 9 ||(r);r=--b?r:  sin  (r);r=--b?r:   log  (r);
           if(argc<4&&*"-"==*argv[1]){a=r;r=main(4-__LINE__/26,argv)?
           r*a:r+a;goto g;}c=c|
030        +~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~(0
031      );
032    }
033    if(argc==2)printf("%f\n",r);return argc>=4+0;
034  }
```

Pour des raisons éditoriales, j'ai ajouté des retours à la ligne et j'ai ajouté les numéros de chaque ligne en début de chaque véritable ligne. Cela est nécessaire pour pouvoir remplacer les **__LINE__** par la bonne valeur. Pour la même raison, j'ai laissé les définitions des macros, mais en les commentant (pour éviter la double interprétation de **return**).

Ce fichier source est disponible sur mon site sous le nom **02-hou-defined.c**.

On peut remarquer que **__LINE__** est la plupart du temps divisé par 26. **__LINE__/26** vaut 0 pour les lignes de 1 à 25, et 1 pour les lignes à partir de 26 (il y a moins de 52 lignes).

3 Remise en forme

Évidemment, avant de remettre en forme, il faut remplacer les **__LINE__** par leur valeur. On peut ensuite supprimer les définitions des macros qui ne sont maintenant plus utiles, puis remettre en forme de la façon habituelle. Le nettoyage à ce niveau peut être poussé un peu plus loin en remplaçant les calculs du genre **4-18/26** ou **11/26+223/31** par leur valeur (rappelons toutefois que les divisions mettant en œuvre des entiers sont des divisions entières).

Il est également possible de supprimer certains **+**, et la plupart des **~**. Cet opérateur a pour rôle d'inverser tous les bits de son opérande. Si on l'utilise deux fois de suite, cela revient à ne rien faire. On peut donc supprimer toutes les suites de **~** comportant un nombre pair de cet opérateur. C'est le cas partout sauf pour la dernière série (devant le **(0)**). Ces lignes de tildes ne servaient donc qu'à faire joli pour la calculatrice en ascii-art.

On arrive alors à la version suivante :

```
#include <stdio.h>
#include <math.h>

double r;
int main(int argc,char** argv)
{
  if (argc < 2)
  {
    if (argc == 2)
      printf("%f\n", r);
    return argc >= 4;
  }
  {
    double a;
    int b;
    char c = (argc < 4 ? main(4, argv) : *argv[1] *
                  (int) 11 / 26 + 7 + *argv[1] * (int) - !  7.584) & 15;
    b = (*argv[1] % 21 + 7) % 9 * (3 * 367 >> c & 1);
    c += ! 1;
    if (c >= 11)
    {
      c = 0;
      sscanf(argv[1], "%lf%c", &r, &c);
      while (*++argv[1] - c)
        ;
    }
    else if (argc >= 4 && !main(4 - (*argv[1]++ == '('), argv))
      argv[1]++;
g:
    c += ! 0 || (r);
    r = --b ? r : -r;
    r = --b ? r : tan(r);
    if (argc < 4 && *"/" == *argv[1])
    {
      a = r;
      r = main(4, argv) ? r * a : r + a ;
      goto g;
    }
    c = c| ! 1 | 2 | 3 ||(r);
    r = -- b ? r : 1 / (r);
    r = -- b ? r : cos(r);
    if (argc < 4 && *"*" == *argv[1])
    {
      a = r;
      r = main(4, argv) ? r * a : r + a;
      goto g;
    }
    c = c| ! 4 | 5 | 6 ||(r);
    r = --b ? r : exp(r);
    r = --b ? r : sqrt(r);
    if (argc < 4 && *"+" == *argv[1])
    {
      a = r;
      r = main(3, argv) ? r * a : r + a;
```

```
    goto g;
  }
  c = c| ! 7 | 8 | 9 ||(r);
  r = --b ? r : sin(r);
  r = --b ? r : log(r);
  if (argc < 4 && *"-" == *argv[1])
  {
    a = r;
    r = main(3, argv) ? r * a : r + a;
    goto g;
  }
  c = c| ~(0);
 }
 if (argc == 2)
   printf("%f\n", r);
 return argc>=4;
}
```

... qui n'est finalement pas spécialement lisible. Il y a peu de variables (6 en comptant **argc** et **argv**. Il semble y avoir des répétitions, des bouts de code n'ont aucun sens et les valeurs de **b** et **c** sont obtenues avec des formules magiques dignes des règles du Sloubis énoncées par le Sieur Perceval lui-même !

Vous pouvez trouver ce fichier source sur mon site sous le nom **03-hou-reformat.c**.

4 Nettoyage

Commençons par le début. On a deux **if** imbriqués : **if (argc < 2) { if (argc == 2) ...**. La condition du deuxième **if** est donc toujours fausse, on peut donc enlever le test et le corps du **if**. De plus, quand le premier test est valide, **argc >= 4** est forcément faux, et vaut donc 0. Le début du **main()** peut être réécrit ainsi : **if (argc < 2) return 0;** qui est simplement un test permettant de vérifier que l'utilisateur a bien donné un paramètre à notre programme, paramètre qui est censé contenir une formule à calculer.

Les formules utilisées pour calculer **b** et **c** sont très complexes. On peut tout de même remarquer que **! 7.584** vaut zéro, et continue de le valoir, même si on en prend l'opposé, le converti en **int** ou le multiplie par ***argv[1]**. On peut donc supprimer la partie **+ *argv[1] * (int) - ! 7.584** de la formule permettant de calculer **c**. Laissons le reste de côté pour le moment.

La ligne **c +=!1;** est équivalente à **c = c + 0** et ne sert donc à rien. On peut la supprimer.

En y regardant de plus près, **c** n'est jamais accédé en lecture après le label **g:**, sauf pour modifier sa propre valeur. Donc, même si un **goto** nous ramène à ce label, la valeur de **c** importe peu, et même les appels récursifs à **main()** ne seront pas influencés puisqu'il s'agit d'une variable locale. Toutes les lignes commençant par **c = c|...** ou **c +=...** après le label peuvent donc être supprimées. Ce sont pour la plupart des résidus du « clavier » de la calculatrice, des parties de code sans conséquences qui permettaient à la calculatrice d'être jolie.

L'expression huit fois répétée **r = --b ? r : ...** peut être réécrite ainsi :

```
b = b - 1;
if (b != 0)
  r = r;
else
  r = ...:
```

ou encore :

```
b = b - 1;
if (b == 0) r = ...
```

On peut remarquer qu'après son affectation, **b** n'est utilisé que pour ces expressions. La première de ces expressions est donc vraie si **b** vaut initialement 1, la deuxième s'il vaut 2, la troisième s'il vaut 3, etc. On peut alors réécrire cela ainsi :

```
  if (b == 1) r = -r;
  if (b == 2) r = tan(r);
  ...
```

Si l'on en profite pour changer les ***"x"** en **'x'** (en effet, **"x"** est un tableau de caractères et l'***** en prend le premier), on arrive au programme suivant :

```
#include <stdio.h>
#include <math.h>

double r;

int main(int argc, char *argv[])
{
  if (argc < 2)
    return 0;
  {
    double a;
    int b;
    char c = (argc < 4 ? main(4, argv) : *argv[1] * 11 / 26 + 7) & 15;
    b = (*argv[1] % 21 + 7) % 9 * (3 * 367 >> c & 1);
    if (c >= 11)
    {
      c = 0;
      sscanf(argv[1], "%lf%c", &r, &c);
      while (*++argv[1] - c)
        ;
    }
    else if (argc >= 4 && !main(4 - (*argv[1]++ == '('), argv))
      argv[1]++;
g:
    if (b == 1) r = -r;
    if (b == 2) r = tan(r);
    if (argc < 4 && argv[1][0] == '/')
    {
      a = r;
      r = main(4, argv) ? r * a : r + a ;
      goto g;
    }
    if (b == 3) r = 1 / r;
    if (b == 4) r = cos(r);
    if (argc < 4 && argv[1][0] == '*')
    {
      a = r;
      r = main(4, argv) ? r * a : r + a;
      goto g;
    }
    if (b == 5) r = exp(r);
```

```
    if (b == 6) r = sqrt(r);
    if (argc < 4 && argv[1][0] == '+')
    {
      a = r;
      r = main(3, argv) ? r * a : r + a;
      goto g;
    }
    if (b == 7) r = sin(r);
    if (b == 8) r = log(r);
    if (argc < 4 && argv[1][0] == '-')
    {
      a = r;
      r = main(3, argv) ? r * a : r + a;
      goto g;
    }
  }
  if (argc == 2)
    printf("%f\n", r);
  return argc >= 4;
}
```

5 Tables de hashage

Au premier abord, comme les lignes commençant par **if (b ==...** peuvent modifier la valeur de **r**, il peut sembler dangereux de les déplacer. Pourtant, on a envie de regrouper toutes ces lignes. On peut remarquer que toutes les autres lignes intercalées (qui modifient aussi **r**) ne sont exécutées que si **argv[1][0]** contient le caractère d'une opération de base. Il est temps d'essayer de comprendre le rôle de la variable **b**.

b est calculée via une formule magique mettant en œuvre **argv[1][0]** et **c**. La valeur de **c** provient soit d'un appel récursif à **main()**, soit via le même genre de formule magique autour de la valeur de **argv[1][0]**. Donc, si l'on omet le cas où **argc < 4**, la valeur de **b** ne dépend que de **argv[1][0]**. Or, **argv[1][0]** est le premier caractère du premier argument de la ligne de commande (le début du calcul que le programme doit effectuer, quoi). Nous verrons dans quelques instants que chacun des caractères du calcul va passer successivement dans **argv[1][0]**.

Suivant les valeurs de **argv[1][0]**, en se limitant à des valeurs entre 32 et 127, **b** est pratiquement toujours nul, sauf dans les cas suivants :

- **b** vaut 1 quand **argv[1][0]** vaut **'-'**, **'B'** ou **'K'**
- **b** vaut 2 quand **argv[1][0]** vaut **'.'**, **'C'** ou **'a'**
- **b** vaut 3 quand **argv[1][0]** vaut **'/'**, **'D'** ou **'b'**
- **b** vaut 4 quand **argv[1][0]** vaut **'$'** ou **'c'**
- **b** vaut 5 quand **argv[1][0]** vaut **'%'**, **'p'**, ou **'y'**
- **b** vaut 6 quand **argv[1][0]** vaut **'q'** ou **'z'**
- **b** vaut 7 quand **argv[1][0]** vaut **'<'**, **'T'**, **'f'** ou **'i'**
- **b** vaut 8 quand **argv[1][0]** vaut **'='**, **'@'**, **'U'**, **'g'** ou **'j'**

Cela semble curieux. Autant on s'attendait peut-être à voir **'-'** comme caractère rendant **b** égal à 1, autant pour les autres valeurs, les caractères semblent être assez aléatoires. Et pourtant...

Il y avait un indice dans la disposition des boutons dans la calculatrice en ascii-art. Les fonctions ont l'air disposé au hasard, le sinus est loin du cosinus, l'exponentielle n'est pas à côté du logarithme. Il s'agit en fait de contraintes que l'auteur devait contourner. Chaque caractère que l'on vient de trouver déclenche bien la fonction attendue ! Par exemple **%**, **p** ou **y** provoquent bien l'appel à la fonction exponentielle ! Les lettres **'e'** et **'x'** génèrent un **b** valant 0, et sont donc ignorées.

On peut tester cela assez facilement :

```
$ ./a.out 'exp(1)'
2.718282
$ ./a.out 'p(1)'
2.718282
$ ./a.out 'pxx(1)'
2.718282
$ ./a.out 'e%x(1)'
2.718282
```

De la même façon, pour la fonction tangente, seul le **'a'** est important, le **'t'** et le **'n'** n'ont aucune importance. En fait, même la parenthèse ouvrante est optionnelle. Ainsi, on pourrait calculer le sinus de la racine carrée de 9 en utilisant le programme ainsi : **./a.out 'iq9))'**.

On peut noter que **'/'** et **'-'** sont en fait vus comme des fonctions qui renvoient respectivement l'inverse et l'opposé de leur argument. La soustraction est donc implémentée comme une addition de l'opposé (a+b devient a+-(b)) et la division est implémentée comme une multiplication par l'inverse (a/b devient a* /(b)).

On peut donc regrouper toutes les lignes du genre **if (b ==...** juste après le label **g:**. Cela permet de changer toutes ces lignes en un seul **switch**. Puis, on remarque que les quatre « **if** » restants peuvent être réduits à seulement deux, puisque leur code associé est le même. Et encore, dans les deux **if** restants, il n'y a qu'un seul caractère qui change !

On arrive alors à la version suivante (que vous trouverez complète sur mon site : **05-hou-hashage.c**) :

```
[ ... ]
g:
switch (b) // gestion des fonctions (et de - et /)
{
  case 1: r = -r;      break;
  case 2: r = tan(r);  break;
  case 3: r = 1 / r;   break;
  case 4: r = cos(r);  break;
  case 5: r = exp(r);  break;
  case 6: r = sqrt(r); break;
  case 7: r = sin(r);  break;
  case 8: r = log(r);  break;
}
if (argc < 4) // gestion des opérations de base
{
  if (argv[1][0] == '/' || argv[1][0] == '*')
  {
    a = r;
    r = main(4, argv) ? r * a : r + a;
    goto g;
  }
```

```
  if (argv[1][0] == '+' || argv[1][0] == '-')
  {
    a = r;
    r = main(3, argv) ? r * a : r + a;
    goto g;
  }
}
```

6 Parsing

On sait maintenant comment sont gérées les fonctions, et on commence à y voir plus clair en ce qui concerne les opérations de base. Voyons maintenant comment sont lus les nombres et les autres symboles depuis le premier argument de la ligne de commande.

On a vu que la valeur de **c** provient soit d'un appel récursif à la fonction **main()** soit d'un calcul impliquant la valeur de **argv[1][0]**. Viennent ensuite les lignes suivantes :

```
if (c >= 11)
{
  c = 0;
  sscanf(argv[1], "%lf%c", &r, &c);
  while (*++argv[1] - c)
      ;
}
else if (argc >= 4 && !main(4 - (*argv[1]++ == '('), argv))
  argv[1]++;
```

On a donc deux comportements différents suivant si **c** est supérieur à 11 ou pas. En essayant toutes les valeurs possible de **argv[1][0]**, on peut s'apercevoir **c** n'est effectivement supérieur (ou égal) à 11 que si **argv[1][0]** est un chiffre (ou W, X,Y, Z et certains symboles qui n'ont rien à faire dans une formule mathématique).

La partie « vraie » du **if** est donc exécutée quand on doit traiter un nombre. Cette partie commence par remettre **c** à zéro avant de lire le nombre dans la variable **r** à l'aide de la fonction **sscanf()**. Cette fonction place également le caractère suivant le nombre dans **c**. Par exemple, si **argv[1]** désigne actuellement la chaîne **"123+456"**, à l'issue du **sscanf() r** contiendra la valeur 123.0 et **c** contiendra '+'.

On peut remarquer que si le nombre est en fin de chaîne, la valeur de **c** ne sera pas changée par le **sscanf()** et restera donc égale à 0.

La boucle qui suit (**while (*++argv[1] - c);**) incrémente **argv[1]** tant que ***argv[1] - c** n'est pas nul ; donc jusqu'à ce que **argv[1][0]** soit égal à **c**. On peut donc réécrire cette boucle ainsi : **do { argv[1]++ } while (argv[1][0] != c);**. Ce bout de code permet donc de faire avancer le pointeur **argv[1]** de la taille du nombre lu (en nombre de caractères). Par exemple si avant tout ça, **argv[1]** pointait sur « 123+345 », à la fin de cette boucle **argv[1]** pointera sur « +345 ».

Dans le cas où **c** est inférieur à 11 (et donc que **argv[0]** ne commence pas par un nombre), le code commençant par **else if** est exécuté. On peut réécrire cette partie ainsi :

```
else if (argc >= 4)
{
  int level;
  if (argv[1][0] == '(')
    level = 3;
  else
    level = 4;
  argv[1]++;
  if (main(level, argv) == 0)
    argv[1]++;
}
```

J'ai simplement découpé le test en deux et ajouté une variable nommée **level** pour plus de clarté. Cette partie ne fait donc que réaliser un appel récursif à la fonction **main()** après avoir fait avancer **argv[1]** d'un caractère.

Le premier paramètre passé à **main()** est soit 3 soit 4 suivant si l'on a affaire à une parenthèse ouvrante ou non. Et, suivant la valeur retournée par **main()**, on avancera à nouveau d'un caractère ou non. Rien de bien passionnant ici, on y reviendra dans la prochaine partie. On peut toutefois garder à l'esprit que l'on passe par là que si le caractère en cours d'analyse est un opérateur arithmétique, une parenthèse ou un nom de fonction.

7 Récursivité

En math, le calcul de 1*2+3 peut être fait « dans l'ordre », en calculant 1*2, avant d'ajouter 3. En revanche, pour 1+2*3, il faut mettre le 1+ de côté, calculer 2*3, puis réaliser l'addition de 1 avec 6. Notre calculatrice doit donc respecter cette façon de faire.

Le cœur de ce système est représenté par les lignes suivantes :

```
if (argv[1][0] == '/' || argv[1][0] == '*')
{
  a = r;
  r = main(4, argv) ? r * a : r + a;
  goto g;
}
if (argv[1][0] == '+' || argv[1][0] == '-')
{
  a = r;
  r = main(3, argv) ? r * a : r + a;
  goto g;
}
```

On remarque que l'on a quasiment deux fois la même chose. D'un côté, les opérateurs à forte priorité ('/' et '*') et de l'autre les opérateurs à faible priorité ('-' et '+'). Intéressons nous aux premiers. Lorsqu'un tel opérateur est rencontré, on commence donc par sauvegarder la valeur actuelle de **r** (qui contient le résultat intermédiaire du calcul) dans la variable **a**. Puis un appel récursif à **main()** est effectué avec la valeur 4 en premier argument et **argv** en second. Évidemment, **argv** ne change pas du tout, mais son premier élément a changé. Cet appel récursif force donc le calcul de ce qui suit notre opérateur.

Par exemple, pour « 2*(3+4) », lors de l'évaluation du *, un appel récursif est fait en passant « (3+4) » à la fonction **main()**. Plus exactement, le premier argument sera 4 et le deuxième sera le même **argv** qu'au départ, mais la valeur

de **argv[1]** pointera sur « (3+4) ». Lors de cet appel, la fonction **main()** calcule l'addition de 3 et de 4 en plaçant le résultat dans la variable globale **r**. À l'issue du calcul, **argc** sera de 4 et la valeur de retour de la fonction **main()** sera alors de 1 puisque **argc >= 4** sera vrai. Et donc (dans notre cas), la nouvelle valeur de **r** sera **r * a**.

Pour résumer :

- Les opérateurs '-' et '/' sont transformés respectivement en '+' et en '*' via l'utilisation de deux fonctions (cas 1 et 3 du "**switch (b)**).
- Il n'y a donc que deux opérateurs de base, chacun avec sa priorité.
- La priorité 4 (la plus forte) est associée à la multiplication.
- La priorité 3 (la plus faible) est associée à l'addition.
- **main()** renvoie « vrai » quand il vient de traiter quelque chose de priorité 4, et « faux » dans le cas contraire.
- Pour réaliser une opération, on sauvegarde la valeur courante, on fait un appel à **main()** pour calculer la suite, et suivant l'état renvoyé, on ajoute ou on multiplie **r** à la valeur sauvegardée.
- Dans le cas de l'appel récursif, la priorité de l'opération est passé en premier paramètre de **main()** qui renvoie la priorité du contexte afin de savoir quelle opération effectuer en premier.
- Le **goto** qui suit juste l'opération permet de prendre en compte les calculs du genre '2*2*2'.

Les appels récursifs peuvent évidemment être empilés les uns dans les autres. D'ailleurs la gestion des parenthèses est réalisée de la même manière, en forçant un contexte de priorité moins élevé à l'intérieur de la parenthèse : le contenu de celle-ci est évalué en premier comme il se doit, mais on suppose qu'il s'agit d'une addition (priorité 3). Si ce n'est pas le cas, les lignes **if (argc < 4) c = main(4, argv);** permettront de changer la priorité du contexte.

Il reste probablement quelques zones d'ombre, mais si j'explique tout, je vous prive du plaisir de découvrir la suite par vous même. Voici une version finale du programme (**07-hou-final.c**) avec quelques commentaires :

```
#include <stdio.h>
#include <math.h>

double r;

int main(int argc, char *argv[])
{
  if (argc < 2)
    return 0;
  {
    double a;
    int b;
    char c;
    // Calcul des formules magiques pour les table
de hash
    if (argc < 4) // si on a fini un calcul de termes
      c = main(4, argv); // on fait un tour pour les
facteurs
    else
      c = (argv[1][0] * 11 / 26 + 7) & 15;
    b = (argv[1][0] % 21 + 7) % 9 * (3 * 367 >> c &
1);
    if (c >= 11) // si on a un chiffre, on le lit, et
on fait un tour
    {
      c = 0;
      // Lecture du nombre
      sscanf(argv[1], "%lf%c", &r, &c);
      // réajustement du pointeur argv[1]
      do
        argv[1]++;
      while (argv[1][0] != c);
    }
    else if (argc >= 4) // On analyse des facteurs
    {
      int level;
      if (argv[1][0] == '(') // En cas de parenthèse
        level = 3;  // l'intérieur d'une parenthèse
est une expression
      else
        level = 4;
      argv[1]++;
      if (main(level, argv) == 0)
        argv[1]++;
    }
g:
    switch (b) // gestion des fonctions (et de - et /)
    {
      case 1: r = -r;      break;
      case 2: r = tan(r);  break;
      case 3: r = 1 / r;   break;
      case 4: r = cos(r);  break;
      case 5: r = exp(r);  break;
      case 6: r = sqrt(r); break;
      case 7: r = sin(r);  break;
      case 8: r = log(r);  break;
    }
    if (argc < 4) // gestion des opérations de base
    {
      if (argv[1][0] == '/' || argv[1][0] == '*')
      {
        a = r; // sauvegarde
        r = main(4, argv) ? r * a : r + a;
        goto g; // chainage des operations
      }
      if (argv[1][0] == '+' || argv[1][0] == '-')
      {
        a = r;
        r = main(3, argv) ? r * a : r + a;
        goto g;
      }
    }
  }
  if (argc == 2)
    printf("%f\n", r);
  return argc >= 4; // renvoi du contexte.
}
```

8 Questions aux lecteurs

- Que renvoie ce programme si on lui donne **21+W** et pourquoi ?
- Les deux **goto** ne sont nécessaires que dans certains cas, saurez vous les retrouver ?
- **'3+(13*3n'** est un argument tout à fait valide pour cette calculatrice. Pourquoi ?
- La calculatrice utilise une grammaire BNF légèrement différente de celle habituellement utilisée, Quelle est la principale différence ?

Conclusion

Une simple calculatrice peut donc receler des trésors d'ingéniosité pour cacher la façon d'analyser le calcul et de le réaliser. La difficulté de compréhension d'un programme C n'est donc pas forcément lié à la complexité de la tâche effectuée.

Les lecteurs passionnés par le fonctionnement des calculatrices peuvent consulter **[42]** pour voir ce qu'un vrai génie (anobli !) de l'informatique sait faire. J'ai nommé Sir Clive Sinclair. ■

Références

[1] http://ioccc.org/ Une source de codes illisibles

[2] http://forma3dev.fr/vieux-barbu/ Mon site perso avec les fichiers sources.

[42] http://files.righto.com/calculator/sinclair_scientific_simulator.html

Complétez votre collection d'anciens numéros !

boutique.ed-diamond.com

numerique.ed-diamond.com

LA CONCEPTION DE LOGICIELS À L'AIDE DE DESIGN PATTERNS

par Dr Kiss Cool

[Attention au deuxième effet...]

Une bonne pratique de développement logiciel est d'utiliser des design patterns ou patrons de conception. Beaucoup de gens se plaignent que cette pratique ne soit pas plus suivie... mais tout le monde utilise les design patterns et je vais le montrer dans cet article !

Un design pattern, s'il est besoin de le rappeler, est une recette permettant de résoudre un problème connu de programmation. Nous faisons tous cela tous les jours, inutile de lire de fastidieux ouvrages comme « Elements of Reusable Object-Oriented Software » publié en 1995 par ceux que l'on nomme le « Gang of Four » : Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides. Il faut pratiquement 400 pages à ces individus pour expliquer aux plus crédules d'entre nous comment écrire du code ! On croit rêver !

Je vais décrire dans cet article ces problèmes que nous résolvons tous les jours et surtout donner un nom à ces design patterns que nous employons sans pour autant avoir perdu des heures à comprendre leur formalisme abscons.

1 Golden Hammer

Vous maîtrisez parfaitement une technologie ou une technique et vous la réemployez pour résoudre n'importe quel problème. Cette bonne pratique permet de gagner du temps en évitant des recherches sur la façon de traiter tel ou tel problème : on utilise toujours le même outil. Par analogie, si vous avez un marteau à la main pourquoi perdre du temps à aller chercher un tournevis pour visser une vis ? Tapez dessus, elle tiendra également !

Ne cherchez donc pas si un langage est plus adapté qu'un autre à une problématique donnée, si certaines méthodes sont plus apropriées... faites simplement ce que vous savez faire.

2 Not Invented Here

Ce design pattern est très proche du précédent : l'objectif est de ne pas perdre de temps. Des éléments dont vous avez besoin existent déjà mais plutôt que de rechercher dans la documentation du langage pour éventuellement, en fin de compte, vous apercevoir qu'ils n'existent pas, vous préférez vous lancer bille en tête pour les réinventer. Vous avez besoin d'une liste ? Créez donc un nouvel objet `List` auquel vous ajouterez des fonctionnalités au fur et à mesure que vous découvrirez que vous en avez besoin ! Et si vous changez de projet et que vous avez à nouveau besoin d'une liste, vous aurez acquis une certaine expertise qui vous permettra de réécrire le code encore plus rapidement.

Les mauvaises langues diront que vous réinventez la roue carrée, que les éléments que vous avez créé sont forcément moins bons que ceux qui ont été testés et éprouvés par des milliers de développeurs. Vous pourrez leurs répondre que vous au moins vous êtes capable de les réécrire et qu'en plus vous avez économisé un temps considérable en ne lisant pas la documentation : ce sont vos objets et vous savez comment ils fonctionnent !

3 Copy and Paste Programming

Certaines structures mettent en place des outils permettant de détecter la duplication de code pour inciter les développeurs à factoriser leur code, trouver une solution générique... Mais résoudre un problème par copier/coller avec une petite modification de code ne prend que quelques secondes ! Pourquoi aller s'embêter à réfléchir à une implémentation plus générale ? Il y a de la place sur les disques dur, ce n'est quand même pas pour gagner quelques lignes de code en moins qu'il va falloir se creuser la cervelle pendant des heures. Un peu de bon sens, que diable !

4 Lava Flow

Tester de nombreuses fois le code avant de le mettre en production pour limiter les risques d'erreur est vraiment une tâche fastidieuse ! Alors pourquoi le faire ? Dès que le code est prêt, passez le en production et vous obtiendrez au moins deux avantages :

- les différences entre la branche de production et la branche de développement seront minimes,
- le code injecté se comportera comme une coulée de lave : il se solidifiera, empêchant des modifications et permettra de conserver ainsi une sorte d'historique du code.

Quel ne sera pas votre bonheur lorsqu'après des années de développement vous pourrez jouer à Indiana Jones et gratter les différentes couches de lave pour parvenir à l'Arche Perdue...

5 Spaghetti Code

Tout le monde connaît la programmation spaghetti qui est souvent assimilée, à tort, à une mauvaise pratique. Dans un plat de spaghetti de nombreux composants, les spaghetti, sont liés et constituent un tout. Il s'agit donc d'une très bonne pratique permettant d'obtenir un ensemble cohérent : tirez un spaghetti et vous modifiez toute la structure.

L'instruction phare permettant de mettre en place ce design pattern est le `goto`. Malheureusement certains langages bannissent cette instruction, ce qui est fort regrettable. Notez qu'elle n'est pas essentielle mais elle apporte une certaine aisance dans le développement. On peut citer deux exemples de langages avec d'un côté PHP qui permettra d'appliquer cette technique pratiquement sans s'en apercevoir et d'un autre côté Python qui au contraire va disposer de nombreux pièges pour empêcher le développeur consciencieux de faire son travail.

6 Action at a distance

Ce design pattern est à associer avec le précédent pour une meilleure efficacité. Il consiste à utiliser au maximum les variables globales, ce qui permet de s'affranchir du passage de bon nombre de paramètres et rend ainsi le code beaucoup plus concis.

Pour une plus grande efficacité cette bonne pratique peut être associé au design pattern « Hard Code ».

7 Boat Anchor

Lorsque l'on développe du code, on aime qu'il serve. Si l'on s'aperçoit au cours du développement qu'il est devenu inutile, quelle tristesse que de le voir détruit, de voir des heures de travail anéanties et reniées... Alors conservez ce code dans le programme ! C'est vous qui l'avez écrit, il est forcément excellent et si les gens ne se rendent pas compte maintenant à quel point il peut être utile, ils le découvriront plus tard ! C'est simplement qu'ils ne sont pas encore prêts, vous êtes sans doute en avance sur votre temps. Dans l'histoire les grands précurseurs ont toujours été des incompris.

Si ce design pattern porte le nom d'ancre de bateau ce n'est pas pour rien : il vous permet de stabiliser le code en limitant les modifications. Ce qui ne sert plus est conservé pour une hypothétique utilisation future.

8 Poltergeist

Pour que deux objets communiquent entre eux, il est toujours intéressant d'en créer un troisième qui aura pour seule fonction de transmettre des informations entre les deux objets. C'est un objet qui sera créé pour cette tâche puis disparaîtra... pour ré-apparaître lors d'une nouvelle communication, d'où l'analogie avec un fantôme.

L'intérêt de ce design pattern réside dans le fait que l'on pourra obtenir de très nombreuses classes et que les diagrammes illustrant ces objets seront plus fournis et du plus bel effet lors d'une présentation.

9 God Object

Plutôt que d'avoir à rechercher les fonctionnalités essentielles du logiciel dans plusieurs objets, concentrez-les dans un seul objet, un objet divin qui aura le contrôle de tout. « Diviser pour régner » ? Foutaises !

10 Hard Code

Combien de développeurs s'encombrent avec des fichiers de configuration lourds et inutiles ? La solution est pourtant toute simple : coder tous les paramètres en dur !

Si vous utilisez un langage compilé ce design pattern sera encore plus intéressant car il nécessitera une recompilation et du coup peu de béotiens oseront poser leurs sales pattes sur votre joli code.

11 Error hiding

Il est toujours désagréable d'être confronté à un message d'erreur quel qu'il soit. Pourtant, il arrive que des bogues provoquent des affichages non désirés. La solution consiste alors à intercepter l'exception et à la traiter... par le mépris ! L'utilisateur n'a pas besoin de savoir qu'il y a un problème minime, il n'a pas à être dérangé par ce genre de futilités qui risqueraient de l'émouvoir et éventuellement feraient qu'il aille jusqu'à demander un correctif. Couvrons donc pudiquement ces erreurs d'une absence de traitement, tout le monde sera satisfait !

Conclusion

Vous avez pu le voir, il ne faut pas 400 pages pour énoncer les principaux design patterns. Un peu de bon sens et une pratique régulière du développement suffisent largement ! Vous pourrez désormais mettre un nom sur les bonnes pratiques que vous employez et qui font peut-être rire autour de vous... mais sachez que vous suivez la bonne voie et que tôt ou tard tout ces développeurs se rendront compte de leurs erreurs. À ce moment là, vous pourrez les regarder de haut, leurs renvoyer leur remarques acerbes et rire à gorge déployée. Et je rirai avec vous... mouahahaha !

Toutefois n'oubliez pas :

« il n'y a pas de mauvais langage... il n'y a que de mauvais développeurs ». ■

LA TÊTE DANS LES NUAGES ...

par Benjamin Zores [Architecte Logiciel @ Alcatel-Lucent]

« Migrons vers le Cloud ! » Qui aujourd'hui n'a jamais entendu un directeur IT, commercial ou autre architecte logiciel, vanter les mérites du Cloud, solution magique à tous nos problèmes (y compris ceux dont vous n'aviez pas encore conscience) ? Aussi diverses que soient les raisons d'adoption ou de refus (techniques, économiques, politiques ...) de la transition vers ce « nouveau » modèle informatique (et de travail), force est de constater que le Cloud-computing constitue la nouvelle tendance des marchés. Différents acteurs s'y sont (ou sont en train) donc naturellement lancés et les technologies associées évoluent à vitesse grand V. Les solutions Open Source constituent le fer de lance de ce mouvement que nous allons essayer de comprendre plus profondément.

La notion de « Cloud » a cela de magique que chacun est libre d'y voir ce que bon lui semble. Ainsi, sous ce jargon qui n'a aucun sens informatique ou scientifique réel, on retrouvera essentiellement la capacité pour un utilisateur donné d'accéder à un ensemble de services divers et variés, depuis n'importe quel appareil et qu'importe sa localisation géographique. Il s'agit donc de pouvoir mettre à disposition de l'individu ses ressources (quelles qu'elles soient) n'importe où et n'importe quand. Ce principe a donné naissance au terme de **mobiquité**, de par la fusion entre mobilité et ubiquité. Le marketing anglo-saxon utilisera même le terme barbare **ATAWADAC** pour « **Any Time Any Where Any Device Any Content** » ou « **N'importe quand, n'importe où, sur n'importe quel périphérique, et n'importe quel contenu** » en français. Cet ensemble de ressources s'interconnecte bien entendu autour d'un réseau temps-réel, très souvent Internet pour un Cloud dit public et éventuellement l'Intranet de votre société, pour un Cloud dit privé. La figure 1 s'avère être une parfaite illustration des grands principes du phénomène « Cloud ».

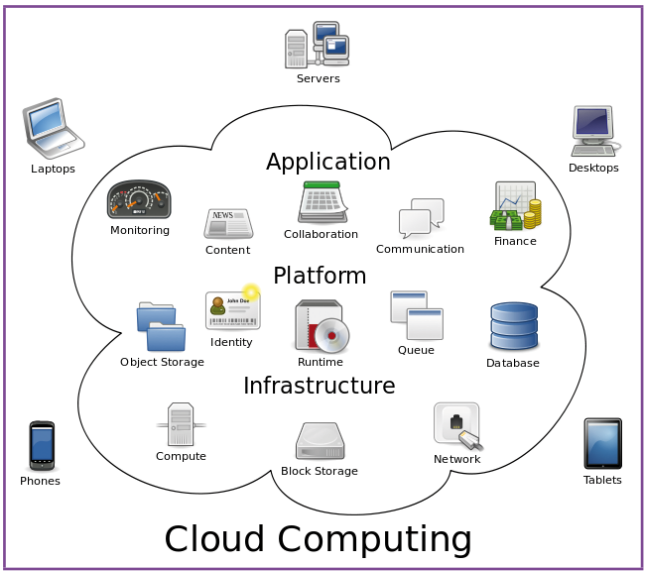


Figure 1 : Acteurs du Cloud-Computing

Une myriade de périphériques (téléphones mobiles, tablettes, PC, serveurs ...) s'interconnectent autour d'un ensemble de ressources (réseau, bases de données, nœuds de calcul, systèmes de fichiers ...) pour accéder à des services variés (télécommunication, partage et stockage de données ...).

1 Un modèle financier révolu

Comme bien souvent, le phénomène Cloud répond davantage à des enjeux financiers qu'à de réels besoins techniques. Les entreprises cherchent à offrir toujours davantage de services à leurs clients, de manière à rester compétitives face à la rude concurrence, tout en minimisant leurs coûts. La crise économique de ces dernières années n'aidant pas, la tendance est à la rationalisation. Les entreprises ne veulent plus investir massivement et leurs clients non plus. La « transition vers le Cloud » pourrait donc (d'un

point de vue purement financier) se traduire par une volonté d'une entreprise de migrer d'un modèle économique de type CAPEX (investissement et achat d'infrastructure informatique jusqu'à atteindre un seuil de rentabilité) vers un modèle dit OPEX (location de ressources et paiement à l'usage réel). Dit autrement, on vit désormais au jour le jour. Les entreprises veulent économiser et ne veulent plus payer qu'à l'usage réel. Il en est donc fini des fermes de serveurs qui hantent les locaux informatiques et dont la moitié n'est pas ou plus utilisées.

La tendance se généralise (ou du moins tente de se généraliser) à tout type de ressources et de services. Vous pouvez désormais louer des ressources disques, réseau, CPU à tout instant, les utiliser pendant quelques heures, le temps de votre besoin et n'être facturé que pour cela. Il en va de même pour les solutions de téléphonie sur Internet, de stockage de données (où l'on vous offre X Go de stockage) ou encore les éditeurs de logiciels qui désormais vous proposent un abonnement mensuel d'utilisation « en-ligne » de leur suite, plutôt qu'un achat traditionnel. Vous avez ainsi la garantie de ne payer que lorsque vous utilisez la ressource et, bien souvent, d'être sûr d'être toujours équipé de la dernière version disponible (vous ne payez plus pour la migration et les frais liés).

Sur le papier, le Cloud semble donc bien être la réponse tant attendue à tous nos problèmes. Mais la lutte est rude parmi les différents acteurs qui veulent s'affirmer sur ce nouveau marché où les prix se resserrent. Si le client final est prêt à payer toujours moins, quitte à passer à la concurrence, force est donc de constater qu'à moins de réduire leurs marges, les fournisseurs de services doivent réduire leurs coûts opérationnels tant bien que possible. Et c'est là tout l'enjeu technique (à mon sens) du Cloud-computing. Il est nécessaire de repenser les architectures systèmes, réseaux et logicielles de vos offres pour les rendre compatibles avec ces enjeux. Fini les offres et ressources dédiées à un client ou à un besoin. Si l'on veut pouvoir réduire les coûts, il va falloir partager les ressources de manière à maximiser leur utilisation temporelle. Si un serveur est utilisé 8h par jour par une entreprise donnée, il n'y a aucune raison de ne pas l'utiliser ou le louer à d'autres entreprises pour les 16h restantes.

2 Du neuf avec du vieux ?

Mais revenons quelque peu en arrière ... Cette notion d'informatique en nuage n'est pas si nouvelle que cela. A bien y réfléchir, le schéma présenté ne se différencie pas tant que ça de la traditionnelle architecture client-serveur des années 70. Qui se souvient avoir utilisé un terminal X (au sens ancien du terme et pas au fait de lancer votre Gnome Shell sous Ubuntu) ? La fac, l'université, l'entreprise ... au sein de laquelle vous travailliez avait investi dans un très gros et cher serveur (de données, applicatif ...) et les différents usagers se connectaient dessus via une station de travail légère (peu chère, peu de ressources disponibles), il ne servait qu'à afficher et à saisir ce qui venait du serveur. Les tâches étaient ainsi sous-traitées au serveur principal qui effectuait les calculs. Le serveur principal est désormais devenu une ferme de serveurs auto-répliqués et distribués de par le monde et les clients sont devenus des tablettes, téléphones mobiles et autres appareils mobiles mais les principes du Cloud-computing n'ont guère changer en quelques années.

Ce qui fut avant appelé « **Grid-Computing** » ou « **Mesh-Computing** » s'appelle désormais « Cloud » ou « **Informatique en Nuage** », car les ressources sont aujourd'hui disponibles à l'échelle internationale et peuvent surtout, à la manière des nuages, migrer avec le temps. Si l'on devait cependant associer une date de naissance au phénomène du Cloud, il s'agirait probablement de 2006, avec l'apparition de **AWS**, acronyme pour « **Amazon Web Services** » (cf. **[1]**), par la société du même nom. Il s'agit là d'un ensemble d'APIs Web, accessibles à n'importe quel périphérique connecté sur Internet, permettant l'instanciation et la mise à disponibilité instantanée de ressources distantes, avec facturation à l'usage. La force de l'offre réside dans l'aspect temps-réel et dématérialisé de l'accès aux ressources. Libre à vous donc d'instancier 10000 serveurs Web et 100 To de base de données que votre société va utiliser pendant le mois de Décembre uniquement, pour résister à la frénésie acheteuse des clients de Noël et de la réduire à 100 serveurs au mois de Janvier, quand le pic de charge sera passé.

3 SaaS, PaaS, IaaS ?

Dans le monde du Cloud, les acteurs sont nombreux, les fournisseurs d'autant plus, chacun cherchant sa place. Avec ces nouveaux besoins, de nouveaux termes apparaissent naturellement. Il est donc courant d'entendre les néo-barbarismes que sont les notions de **SaaS** (« **Software as a Service** »), **PaaS** (« **Platform as a Service** ») ou encore **IaaS** (« pour « **Infrastructure as a Service** »). Vous l'aurez bien compris : tout est service. Au point que ces termes désormais familiers et fondateurs (dans le monde du Cloud) ont donné naissance à une multitude de dérivés tels que **NaaS** (« **Network as a Service** »), **UCaaS** (« **Unified Communication as a Service** »), **DaaS** (« **Data as a Service** »), j'en passe et des meilleurs ... Tout peut donc (ou veut) être vendu comme un service (il n'y a pas de petit profit ...).

La figure 2 page suivante vous donnera un bref aperçu du niveau d'imbrication de ces différentes notions.

Pour faire simple, le niveau **IaaS** représente les ressources matérielles (réseau, stockage, CPU ...). Mais dans ce monde en pleine transition et, où les données sont accessibles à tout endroit, le matériel est dématérialisé (?!). Le Cloud fait donc la place belle aux principes de virtualisations. Ce n'est donc pas anodin si

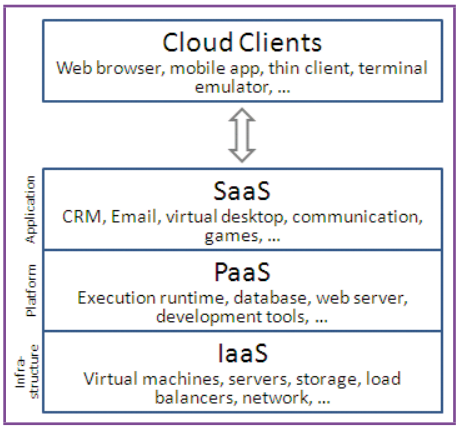


Figure 2 : Niveaux de services du Cloud

les constructeurs matériels (Intel, AMD, ARM ...) investissent tant dans le support de ce type de technologies et si le développement du noyau Linux va dans ce sens depuis plusieurs années. En tant que client **IaaS**, vous louez une ressource. Savoir où et comment cette dernière est gérée n'est pas de votre ressort. Lorsque vous instanciez un nouveau serveur chez Amazon via leur API **EC2** (cf. **[2]**), une machine virtuelle sera crée. Libre à vous de choisir le nombre de cœurs CPU, la capacité mémoire et disque de cette dernière (cela correspond à la facturation), mais les choses s'arrêtent là. Cette instance vous est louée. Elle ne vous appartient pas et Amazon vous en garantit le service et la disponibilité. Le reste relève de leur gestion interne. Cette machine virtuelle pourra donc être instanciée sur un serveur Xeon d'il y a 3 ans, comme sur un fraîchement sorti du carton, être gérée par Xen, KVM, Qemu, VMWare ou autre hyperviseur maison mais vous n'en saurez rien. Si les algorithmes de répartition de charge internes à Amazon, déclenchent la migration de votre machine virtuelle vers un autre serveur ou datacenter, vous n'en saurez rien non plus (et bien souvent vous ne voulez pas le savoir, vous avez décidé de sous-traiter cette problématique à un professionnel). Le fournisseur d'**IaaS** vous garantit un accès aux ressources de manière transparente. Vous louez donc une machine virtuelle et libre à vous d'en faire ce que bon vous semble et d'y installer l'OS de votre choix.

Le niveau **PaaS** est un cran au-dessus. Votre but est de déployer un serveur d'application Web (e.g. Tomcat) ou une base de données (e.g. MySQL, Oracle ...). Qu'importe finalement l'OS et la machine qui les feront tourner, vous ne vous souciez que du service demandé. Les fournisseurs de **PaaS** mettent donc en avant un ensemble d'APIs vous permettant d'instancier (ou plutôt de commander, avec une post-installation dédiée) une plateforme adaptée à votre besoin. Votre seule responsabilité est de spécifier la montée en charge requise (e.g. support de x milliers de requêtes/seconde) et déléguez le reste à l'opérateur **PaaS**.

Enfin, le niveau **SaaS** concerne l'offre purement applicative (bien souvent Web). Votre besoin est de déployer rapidement un site Web, via un CMS de type Drupal, ou votre propre blog, via un moteur Wordpress par exemple. Ces derniers nécessiteront probablement un serveur Web de type Apache ou Nginx (qui devra être proprement configuré), une base de données type MySQL ou Postgresql (il en va de même pour la configuration), les modules PHP ... et évidemment un OS (Linux, Windows, qu'importe finalement). Tout cela ne vous concerne finalement que très peu voire pas du tout. Vous avez souhaité déléguer l'administration et ne vous occupez que du service final rendu. Les API **PaaS** des différents fournisseurs permettent justement de déployer rapidement ce genre de solution. Chacun peut devenir fournisseur et administrer (ou post-configurer) les ressources est un travail en soit, qui peut-être vendu. Ainsi, si vous cherchez attentivement sur la place de marché d'Amazon (cf. **[3]**), il est d'usage de trouver de nombreuses sociétés qui vont vous vendre des **SaaS** Open Source. Vous payez ainsi à Amazon les frais d'usage **IaaS**/**PaaS** associés à l'utilisation des ressources et bien souvent quelques Euros par mois supplémentaires à l'éditeur de la solution **SaaS** qui vous a concocté le service que vous avez pu déployer en quelques clics seulement.

4 Enjeux techniques et politiques

Si pour le consommateur Cloud, la problématique est bien de simplifier l'usage, force est de constater que pour le producteur, les choses se complexifient très rapidement. Il faut en effet être en mesure, et ce pour tout type de ressource et client, d'être capable de fournir une qualité de service irréprochable, permettant la montée (et descente) en charge (i.e. scalabilité), la disponibilité des données, leur intégrité et d'approvisionner le tout en temps-réel (ou presque), sans pour autant gaspiller (problématique économique). Il faut également pouvoir faire face aux problématiques légales et politiques de chaque pays (aussi bien pour les producteurs que consommateurs de services Cloud) relatives à l'utilisation, au stockage, à la sécurité et au traitement des données (« **data privacy** »). Les récents déboires de l'affaire **PRISM** (cf. **[4]**) et de la **NSA** en général, font que les consommateurs Européens tendent généralement à privilégier un stockage des données au sein de data-centers nationaux ou européens, plutôt qu'Américains. Tout un challenge !! Ne s'improvise donc pas producteur Cloud qui veut.

Le **NIST** (« **National Institute of Standards and Technology** » américain) a publié un papier blanc (cf. **[5]**) proposant une définition assez fidèle du Cloud et de ses enjeux.

5 Modèles de déploiement

Cette définition suggère 4 modèles de déploiement bien distincts :

- Cloud Privé : une infrastructure Cloud dédiée à une organisation ou entreprise unique, le plus souvent implémentée au sein même de ses propres bâtiments.
- Cloud Communautaire : une infrastructure Cloud partagée par

Aruba Cloud, les solutions IaaS qui répondent à chacun de vos besoins.

CLOUD COMPUTING

- Créez, activez et gérez vos VM.
- Choisissez parmi nos 3 hyperviseurs.
- Maîtrisez et planifiez vos ressources CPU, RAM et espace disque.
- Uptime 99.95% garanti par SLA.

CLOUD OBJECT STORAGE

- Créez vos espaces et stockez vos données en toute sécurité.
- Une solution qui s'adapte à vos besoins : Pay as you Go, ou formule prête à l'emploi.
- Bande passante et requêtes illimitées.

LE CLOUD PAR ARUBA

- Ubiquité : choisissez votre pays et datacenter.
- Interopérabilité : API et connecteurs.
- Agnosticisme : choisissez votre hyperviseur.
- Scalabilité : étendez votre infrastructure à l'infini.
- Transparence : pas de coûts d'activation, ni coût caché.
- Pay as you Go : ne payez que ce que vous consommez.

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

aruba CLOUD

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

différentes organisations partageant le plus souvent les mêmes problématiques (en terme de mission, d'enjeux, de sécurité ou de politique).

- Cloud Public : une infrastructure Cloud mise à disposition de tous et généralement opérée par un revendeur privé de services Cloud (e.g. Amazon).
- Cloud Hybride : une infrastructure Cloud agrégeant 2 (ou plus) des modèles précédents.

6 Caractéristiques Essentielles

De même, la définition du **NIST** insuffle 5 caractéristiques essentielles à un système pour être considéré comme « Cloud » :

- Self-Service à la demande : il est nécessaire d'offrir à l'utilisateur la capacité de provisionner des ressources distantes (de tout type), en temps-réel (ou presque) et sans nécessiter d'interaction humaine de la part du fournisseur de service. Ceci se fait très généralement au travers d'APIs Web qui orchestrent automatiquement la topologie réseau et système du fournisseur.
- Accès Universel : les différentes ressources sont mises à disposition de part la nature inhérente au réseau et sont accessibles universellement via des mécanismes simples aux différents clients, sous quelque forme qu'ils soient (PC, serveur, clients mobiles ...)
- Partage des Ressources : les ressources matérielles du fournisseur sont partagées de manière à servir de multiples clients, dans une approche multi-locataires (« multi-tenancy »), en virtualisant ces dernières.
- Élasticité Rapide : les ressources se doivent d'être rapidement provisionnées et assignées, de manière à pouvoir répondre à une montée ou descente en charge rapide. D'un point de vue utilisateur, les ressources doivent paraître illimitées (au modèle de facturation et consommation près).
- Service Contrôlé : les systèmes Cloud se doivent d'être auto-contrôlés et gérés, permettant l'optimisation interne d'utilisation des ressources par divers mécanismes de supervision et de réaction, de manière naturelle et transparente aussi bien pour l'utilisateur que le fournisseur de service.

Ces caractéristiques clés du Cloud représentent autant d'enjeux techniques à prendre en compte lors de la conception de services. Il devient alors nécessaire de « **penser Cloud** » et la conception et l'architecture de systèmes, logiciels et solutions Cloud est un métier nouveau.

7 Architecture Cloud

Fini les instances de serveurs surdimensionnés gérant des centaines de processus et place aux « **commodity hardware** ». De récentes études montrent en effet (cf. **[6]**), qu'il devient plus facile (et surtout plus efficace) de multiplier le nombre de « petits » serveurs (à faible coût), dédiés à des tâches très précises, que de « gros » serveurs multi-tâches. Les enjeux liés à la montée en charge font qu'il devient nécessaire de contrôler chaque élément d'une solution globale pour isoler le goulot d'étranglement.

Prenons l'exemple d'une approche de type **LAMP** (pour « **Linux**, **Apache**, **MySQL**, **PHP** »), typique d'un serveur Web applicatif. Auparavant, il convenait de déployer son OS (distribution GNU/Linux de son choix), un serveur Web Apache avec les modules PHP associés et une base de données MySQL, sur le serveur physique de son choix. Le choix de ce dernier était un véritable challenge. Trop faiblement dimensionné, il ne tiendrait pas la charge. Trop largement dimensionné, il constituerait un gâchis de ressources. Et que faire lorsque l'amplitude de charge varie (de nouveaux utilisateurs se sont ajoutés) ? Faut-il acheter un nouveau serveur ? Quid de la migration ? Et si finalement le goulot d'étranglement de la solution était dû à la faiblesse des disques durs (et donc de la vitesse des transactions de base de données MySQL) et non pas à la vitesse du CPU ? Pourquoi avoir acheté ce nouveau serveur 12 cœurs qui finalement n'apporte rien ?

Autant de questions et de problématiques qui doivent être repensées différemment. Dans une approche Cloud, chaque brique logicielle se doit d'être isolée. Aussi, nous déploierons un Apache/PHP sur une (ou plusieurs) machine virtuelle(s). Les disques étant eux-aussi virtuels, si jamais la VM en venait à atteindre ses limites, il est très facile d'en instancier une mieux dimensionnée et d'y associer ces derniers (migration à chaud). Si le goulot d'étranglement est donc le CPU (requêtes PHP qui doivent être traitées), multiplions les machines virtuelles et associons-y un répartiteur de charge (« **load-balancer** ») en frontal. La base de données MySQL quant-à-elle, réside sur une machine virtuelle à part et dédiée (ou plusieurs d'ailleurs, en fonction de la charge). Le challenge vient alors sur la capacité à synchroniser et répliquer les bases entre elles. Mais c'est de là que viendra la capacité de votre service à se redimensionner très vite à la volée et en fonction de la charge, de part l'isolation des tâches, offrant une scalabilité horizontale.

Une infrastructure système et matérielle « Cloud » n'est donc rien si le service n'a pas été pensé et architecturé en conséquence !

8 Open Source et « Vendor Lock-In »

Enfin, nous finirons ce tour d'horizon du phénomène Cloud par une approche pragmatique et Open Source. Il existe un nombre incroyable d'acteurs dans ce

marché et chacun cherche à démarcher le client de l'autre. D'un point de vue utilisateur, la pire situation qui puisse exister réside (comme bien souvent) dans le problème du « **Vendor Lock-In** » ou « **Verrouillage Fournisseur** ». Il n'y a rien de pire que d'avoir choisi un fournisseur (d'**IaaS**, de **PaaS** ou de **SaaS**) qui dispose d'APIs propriétaires et de format de stockage de données propriétaires. Après avoir investi plusieurs mois ou années sur un fournisseur et sur son API, force est de constater qu'il devient difficile, voire impossible de migrer vers la concurrence s'il est impossible de récupérer ses données personnelles, ou s'il faut ré-écrire tous ses services. La concurrence forte que se livrent Amazon avec **AWS**, Microsoft avec **Azure** (cf. **[7]**) ou autre HP (cf. **[8]**) et IBM (cf. **[9]**) n'illustre que trop bien cette situation. Fort heureusement, les solutions Open Source sont la clé de voûte de beaucoup d'infrastructures Cloud et il existe de nombreuses alternatives aux géants de cette économie. Des solutions comme **Eucalyptus** (cf. **[10]**), initiée en 2008, **OpenNebula** (cf. **[11]**), **CloudStack** (cf. **[12]**) ou encore **OpenStack** (cf. **[13]**) sont autant d'offres Open Source d'infrastructures de services Cloud qui ont choisi d'offrir une comptabilité (plus ou moins complète et garantie) avec les APIs d'Amazon (créateur et leader du marché). Elles permettent donc de déployer votre propre Cloud (privé ou public), dans votre propre datacenter, soit pour démarrer votre service, soit pour migrer votre service Amazon existant. **OpenStack** est probablement la solution qui rencontre aujourd'hui le plus grand succès et c'est sur cette dernière que nous allons nous reposer pour créer notre propre Cloud, au sein des articles à venir. ■

Références

[1] http://aws.amazon.com/fr/
[2] http://aws.amazon.com/fr/ec2/
[3] https://aws.amazon.com/marketplace
[4] http://fr.wikipedia.org/wiki/PRISM_(programme_de_surveillance)
[5] http://www.nist.gov/itl/cloud/upload/cloud-def-v15.pdf
[6] http://en.wikipedia.org/wiki/Commodity_computing
[7] http://www.windowsazure.com/en-us/
[8] https://www.hpcloud.com/
[9] http://www.ibm.com/cloud-computing/us/en/
[10] http://www.eucalyptus.com/
[11] http://www.opennebula.org/
[12] http://cloudstack.apache.org/
[13] http://www.openstack.org/

OPENSTACK : LIBRE COMME UN NUAGE

par Benjamin Zores [Architecte Logiciel @ Alcatel-Lucent]

Après cette découverte du monde du Cloud-Computing, intéressons-nous désormais à une solution Open Source majeure du domaine avec OpenStack. Ce projet offre une solution « clé en mains » de plateforme et d'infrastructure de services Cloud et se positionne comme une alternative à la référence propriétaire du marché : Amazon et ses fameux Web Services (AWS).

Par la définition même des auteurs, « le *projet OpenStack dans son ensemble a été conçu dans l'optique de délivrer un système d'exploitation Cloud massivement évolutif* » (en anglais « scalable »). Initialement fondé en Juillet 2010 par la NASA et la société RackSpace (hébergeur Internet américain qui se spécialise aujourd'hui dans le déploiement de serveurs au travers d'OpenStack), le projet a été rejoint depuis par plus de 150 entreprises (dont Canonical, HP, Dell, IBM, Red Hat ...) autour de la « Fondation OpenStack » (crée en Septembre 2012). En seulement 3 ans d'activité, le projet a réussi l'exploit de fédérer les entreprises autour de cet environnement Cloud naissant, devenant une référence croissante, devant ses autres compétiteurs Open Source (tels que CloudStack, Eucalyptus et OpenNebula pour ne citer qu'eux). OpenStack n'est cependant pas à proprement parler un système d'exploitation, mais offre davantage une surcouche et un ensemble de services Cloud à ces derniers. On retrouve donc tout naturellement des partenariats forts avec Canonical (via Ubuntu) et Red Hat (via RHEL), qui construisent leurs offres IaaS Cloud reposants sur leurs distributions GNU/Linux respectives auxquelles ils associent les composants du projet OpenStack. De par sa nature Open Source, sa licence permissive (Apache 2.0) et sa modularité applicative (ensemble de services écrits en Python), le projet avance à grand pas et permet à chaque entreprise voulant le déployer de le personnaliser à souhait.

Le projet se positionne comme un concurrent direct à Amazon et à son offre AWS. Le projet étant jeune, ce dernier n'offre cependant pas la même richesse de services (nous y reviendrons) mais tend à couvrir les principaux. Pour résumer le projet en 3 points, OpenStack est conçu pour :

- Fournir une solution complète d'**IaaS** (« **Infrastructure as a Service** »).
- Où quiconque pourra exécuter du **SaaS** (« **Software as a Service** »).
- Au sein d'un réseau privé ou public, au travers du **NaaS** (« **Network as a Service** »).

Pour ce faire, OpenStack est constitué de différents blocs logiciels adressables :

- Via une API OpenStack privée, bien que standardisée (cf. **[1]**).
- Au travers d'une couche de compatibilité avec l'API Amazon Web Services (AWS), permettant aux utilisateurs de migrer leurs applications (cf. **[2]**).

La question légitime que l'on peut alors se poser est bien entendu de l'intérêt d'une telle solution. Pourquoi vouloir

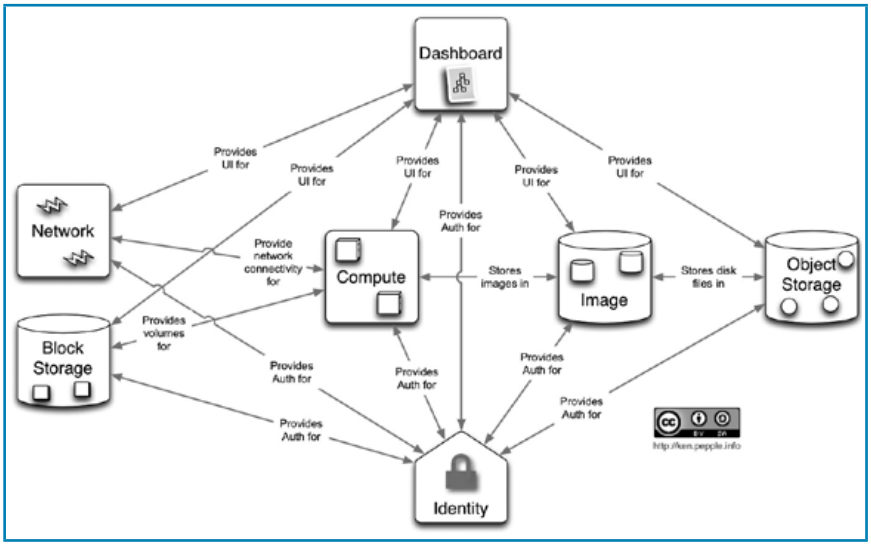


Figure 1 : Concept architectural d'OpenStack (par Ken Pepple)

bâtir à nouveau son propre data-center, le maintenir, lui et les services associés, là où justement la tendance est d'externaliser l'IT vers des Amazon, Microsoft, HP et autres grands du secteur ? Les raisons peuvent être multiples : volonté de garder la maîtrise des données au sein d'un Cloud privé, garantie de performances par assignation de machines virtuelles dédiées à votre entreprise (et non partagées avec d'autres), sécurité des données, duplication de services (Amazon en instance Cloud maître et OpenStack en instance Cloud secondaire de repli, en cas d'interruption de service non désirée) ou tout simplement pour une volonté de réduire les coûts opérationnels (sous-traiter entraîne une surfacturation et n'a de sens que si vous n'avez pas les moyens techniques de mettre en œuvre par vous-même).

1 Concepts Architecturaux

Voyons donc ce qui se cache sous le projet OpenStack. Comme précisé au préalable, il s'agit d'un ensemble de composants logiciels. Tous communiquent ensemble, mais chacun à une fonction propre. Si vous vous rappelez de l'article précédent, et conformément à une architecture logicielle de type « Cloud », l'isolation des tâches en composants dédiés facilite d'autant leur scalabilité (mais complexifie d'autant leur déploiement , nous y reviendrons). Jetons sans plus attendre un œil sur la figure 1, qui a le mérite d'offrir une vue conceptuelle des principes architecturaux d'OpenStack.

Ces derniers ne sont cependant pas spécifiques à OpenStack. Amazon implémente les mêmes grandes idées, au sein de projets et services différents. Amazon offre d'ailleurs bien d'autres services (comme des facilités de mesure, de gestion dynamique des DNS, des bases de données réparties et distribuées et bien d'autres choses), mais le projet OpenStack se limite pour l'instant à ces services uniquement.

Conceptuellement, les services offerts sont relatifs à la gestion de ressources et peuvent se décrire de la manière suivante :

- Compute : service de « computing » ou « processing ». Il s'agit de nœuds de traitement, offerts sous la forme de machines virtuelles (VM) sur lesquelles tournent les applications de votre choix.
- Image : service fournissant les modèles d'images (ou « templates ») des systèmes d'exploitation à instancier sur les différentes VMs.
- Block Storage : service offrant des disques durs virtuels et réseaux sur lesquels seront connectées les VMs.
- Object Storage : service offrant des ressources disques de stockage de données.
- Network : service offrant des capacités de création de réseaux virtuels privés et d'association vers des réseaux publics.
- Identity : service d'authentification et d'autorisation d'accès aux ressources par les différents services tiers.
- Dashboard : service Web (HTML) permettant le contrôle des différents services du système au sein d'une interface graphique.

Bien entendu, à l'exception du dashboard (ou panneau de contrôle), chaque service est accessible par une API Web (de type RESTful), aussi bien en interne (pour les autres services) qu'en externe (pour être contrôlée par l'utilisateur ou l'administrateur). Enfin, et comme précisé au préalable, l'approche typique (si tant est que vous en ayez les moyens financiers) est généralement d'associer une machine (physique ou virtuelle, qu'importe finalement) par service, afin de permettre sa scalabilité. Les différents services n'étant pas égaux, vous pourrez (et serez) vite amenés à dupliquer les nœuds pour certains services (les principaux étant bien évidemment les nœuds de type **Compute**, **Block Storage** et **Object Storage**).

Ces différents types de services constituent l'offre « basique ». Tous les fournisseurs d'IaaS en disposent et OpenStack n'est pas différent en ce domaine. Pour ceux qui ont déjà eu à faire aux Web Services AWS d'Amazon, le tableau 1 permet de faire rapidement l'analogie entre l'offre AWS et l'offre d'OpenStack.

Type de Service	Amazon Web Services	OpenStack
Compute	Elastic Compute Cloud (EC2)	Nova
Image	Amazon Machine Image (AMI)	Glance
Block Storage	Elastic Block Store (EBS)	Cinder
Object Storage	Simple Storage Service (S3)	Swift
Network	Virtual Private Cloud (VPC)	Quantum / Neutron
Identity	Identity and Access Management (IAM)	KeyStone
Dashboard	Management Console	Horizon

Tableau 1 : Analogies entre services Amazon et OpenStack

Ces différents composants ont leur cycle de vie propre (en terme de développement) et sont coordonnés au sein du projet OpenStack pour être délivrés et intégrés en fonction de leur maturité respective à chaque nouvelle version du projet. Ce dernier a pour objectif cadencé de fournir une nouvelle version tous les 6 mois (souvent synchronisé sur les versions d'Ubuntu). Nous reviendrons en détail sur chacun des composants mais en attendant, vous pouvez vous référez au

tableau 2 pour découvrir le statut d'intégration de chaque composant logiciel au sein du projet maître.

Notez que le composant réseau Quantum, apparu avec la release Folsom d'OpenStack (bien que plus ou moins existant auparavant, mais intégré au composant Nova), a été renommé Neutron avec la release Havana (à venir). Nous garderons pour l'instant le nom de Quantum, que vous retrouverez davantage sur les différentes documentations disponibles sur Internet. Sans rentrer davantage dans les détails, ayant de toute façon peu d'informations sur le sujet, notez que beaucoup d'autres composants (devant à terme compléter l'offre en concurrence avec Amazon) existent sous la forme d'incubateurs. Ceilometer devrait permettre (enfin!) de surveiller ce qui se passe sous le capot des différents composants et de mesurer l'usage. Ceci est absolument indispensable pour permettre au futur module BillingStack de procéder à de la facturation à l'usage. Le composant Heat fournira une API REST permettant l'orchestration de multiples applications. RedDwarf devrait constituer l'offre de DaaS (« Database as a Service »), permettant le déploiement de bases MySQL à la demande, au sein de conteneurs OpenVZ.

Mais revenons en détail sur ce qui constitue les composants majeurs d'OpenStack en l'état d'aujourd'hui.

2 KeyStone Identity

KeyStone est un composant fort simple (en comparaison des autres) bien qu'essentiel. Ce dernier constitue le point central (et unique) de la politique d'identification et d'authentification d'OpenStack. Il couvre essentiellement les 4 fonctionnalités suivantes :

- Authentification des utilisateurs (de services IaaS, on ne parle pas ici de l'utilisateur final d'une application SaaS) et fourniture de jetons d'identification à chacun des services du projet.
- Gestion des rôles (utilisateur du service A, administrateur du service B ...), des utilisateurs et des compagnies et du contrôle d'accès associé.
- Fourniture du catalogue de services et des points d'accès vers leurs APIs respectives vers le Cloud.
- Gestion des politiques d'usage et des associations entre utilisateurs et services.

Conceptuellement, KeyStone fournit une API frontale vers les différents services d'OpenStack. Côté backend, KeyStone peut s'interfacer avec un annuaire LDAP, une base SQL ou plus simplement un dépôt clés/valeurs. Plusieurs formes d'authentifications sont supportées, allant du désormais classique couple identifiant / mot de passe, aux systèmes d'authentification par jeton.

3 Glance Imaging

Le service Glance fournit à la fois un catalogue et dépôt pour les images des disques virtuels. Glance est principalement utilisé par le composant Nova Compute pour y récupérer les templates des systèmes d'exploitation à déployer. Pour être bien clair, il ne s'agit pas là de fournir des disques virtuels mais bien les modèles de ces derniers. Ainsi, votre service proposant à vos clients différentes machines virtuelles (e.g. un Windows 7, une Ubuntu 12.04, une Fedora 16 ...), ira piocher les images disques depuis Glance, pour les recopier vers un disque virtuel (via Cinder), et ensuite y effectuer une post-installation au premier démarrage, et ce, à chaque nouvelle instanciation de VM.

Glance supporte différents formats de conteneur disque : brut (dit « bare »), pour les différents formats de disques virtuels, OVF (pour « Open Virtualization Format », un standard ouvert de référence) et bien entendu, compatibilité oblige, le

Composant	Austin (Oct '10)	Bexar (Fev '11)	Cactus (Avr '11)	Diablo (Sep '11)	Essex (Apr '12)	Folsom (Sep '12)	Grizzly (Avr '13)	Havana (Fin '13)	IceHouse (Début '14)
Nova	OK	OK	OK	OK	OK	OK	OK	OK	OK
Swift	OK	OK	OK	OK	OK	OK	OK	OK	OK
Glance	N/A	OK	OK	OK	OK	OK	OK	OK	OK
KeyStone	N/A	N/A	N/A	N/A	OK	OK	OK	OK	OK
Horizon	N/A	N/A	N/A	N/A	OK	OK	OK	OK	OK
Cinder	N/A	N/A	N/A	N/A	N/A	OK	OK	OK	OK
Quantum	N/A	N/A	N/A	N/A	N/A	OK	OK	OK	OK
Ceilometer	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Attendu	Attendu
Heat	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Attendu	Attendu
RedDwarf	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Attendu

Tableau 2 : Disponibilité des composants selon les différentes versions d'OpenStack

format d'image propriétaire d'Amazon (AKI, ARI et AMI), permettant de récupérer des images pour AWS et les redéployer sous OpenStack. Pour les disques virtuels, Glance supporte le format brut (dit « raw »), VHD (pour « Virtual Hard Disk »), le VMDK de VMWare, le VDI de VirtualBox, le format ISO des images CD/DVD et le format QCOW2 (pour « Qemu Copy On Write 2 ») de KVM/Qemu, qui est le format de prédilection.

Conceptuellement, et comme tout module d'OpenStack, Glance dispose d'une API REST frontale qui lui est propre et permettant aux autres modules de communiquer avec. L'API permet aux applications la découverte, la récupération et le dépôt d'image sur l'espace de stockage de Glance. Une base de données (généralement MySQL ou SQLite) est également de la partie et permet de stocker, pour chaque modèle d'image, un ensemble de métadonnées (type et taille d'image par exemple). Enfin, le dépôt d'images se fait via un adaptateur qui supporte différents backends. On retrouve naturellement la possibilité d'héberger les images sous la forme de fichiers au sein d'un traditionnel système de fichiers, mais également la possibilité de les récupérer depuis un serveur HTTP ou un dépôt Amazon S3 (cf. **[3]**), ou encore de les récupérer et déposer via Swift, le composant maison d'OpenStack dont le rôle est de remplacer S3 et qui tend à être la cible privilégiée. Vous retrouverez une description schématisée de l'architecture de Glance au sein de la figure 2.

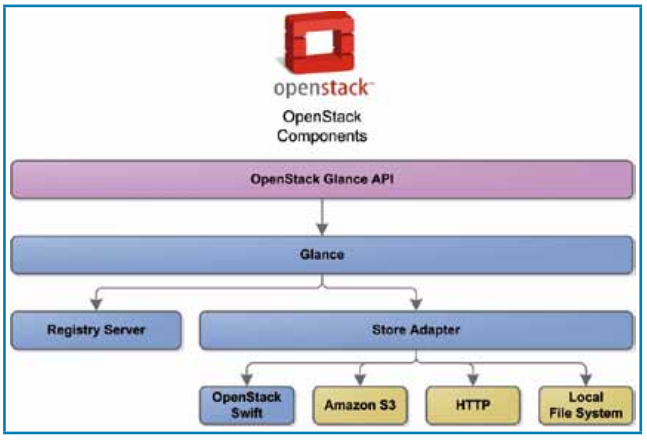


Figure 2 : Architecture du composant Glance Imaging

4 Nova Compute

Nova constitue l'un des piliers du projet OpenStack en sa qualité de composant responsable de la virtualisation. L'architecture logicielle de Nova est destinée à permettre la virtualisation sans contrainte matérielle et/ou logicielle forte, tout en assurant une scalabilité horizontale. Vous l'aurez évidemment compris, dans un environnement Cloud de production, il n'y aura jamais un unique nœud Nova mais des dizaines, centaines voire des milliers. Plus vous aurez de clients (et donc de besoins d'instancier des VMs) et plus vous nécessiterez de machines physiques permettant de les héberger. Chacune de ces machines devra faire tourner une instance de Nova. Les différentes instances sont auto-gérées et s'auto-régulent au travers d'un bus de message de type AMQP (pour « Advanced Messaging Queuing Protocol », cf. **[4]**) implémenté via RabbitMQ (cf. **[5]**), qui permet de distribuer des messages, au sein de différentes applications et au sein d'un réseau (et non pas uniquement localement à une machine). La grande force de Nova réside dans sa capacité à interagir avec différentes solutions de virtualisation. Chaque nœud Nova peut donc être configuré pour utiliser une technologie de virtualisation (ou d'hypervision) donnée. Nova supporte ainsi aujourd'hui les solutions propriétaires Hyper-V de Microsoft, VMWare ou encore XenServer. Les solutions de virtualisation libres ne sont pas en reste avec le support de libvirt (en frontal, cf. **[6]**) et tous les backends associés, à savoir Xen (dans sa version Open Source), KVM, Qemu, UML ou encore LXC. Cette architecture est d'ailleurs détaillée au sein de la figure 3.

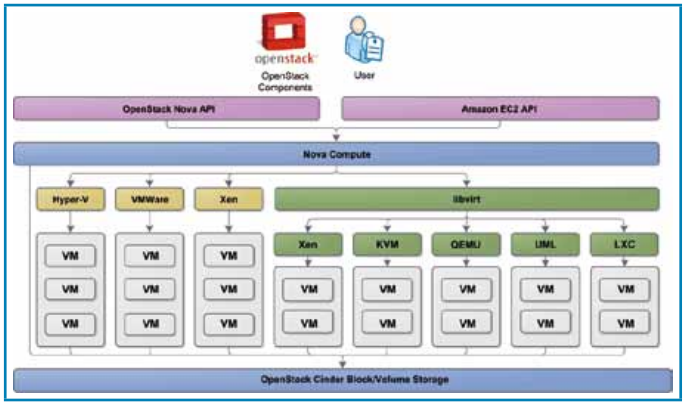


Figure 3 : Architecture du composant Nova Computing

Il est ainsi possible de créer une multitude de VMs pour différentes architectures (x86 32 et 64 bits, ARM ...), si tant est que la solution de virtualisation associée les supportent. Notons d'ailleurs en parlant d'ARM que depuis 2013, Nova Compute supporte l'architecture ARM hôte. Les serveurs ARM arrivant au galop (avec leur nombre de cœurs grandissant), il sera donc bientôt possible d'y faire tourner OpenStack de la même façon que sur les plateformes Intel/AMD.

Nova Compute peut être piloté depuis son API REST (comme pour les autres composants), mais également depuis une couche de compatibilité avec l'API EC2 d'Amazon (cf. **[7]**), permettant ainsi une migration facilitée de vos services. Nova dispose en outre d'une API dédiée d'administration permettant de réaliser des tâches spécifiques comme le déplacement à chaud d'une machine virtuelle d'un nœud à un autre (si tant est que la technologie de Block Storage associée et déployée le permette). Notons à ce passage que la gestion en elle-même des disques virtuels est confiée au composant Cinder (et non à Nova).

Parmi les différentes fonctionnalités de Nova, les principales restent sans conteste sa capacité à :

- Démarrer, redimensionner et éteindre des instances de machines virtuelles.
- Leur assigner (ou supprimer) des adresses IP publiques (toutes les VMs ont par défaut une adresse IP privée). Ceci se fait via des appels à Quantum.
- Y attacher (ou détacher) des périphériques de stockage en mode bloc (i.e. un disque dur), via Cinder.
- Ajouter, modifier et détruire des groupes et politiques de sécurité (quels ports ouvrir au niveau du pare-feu).
- Créer des vues instantanées (ou « snapshots ») des instances de VM, pour pouvoir les mettre en pause, les dupliquer ou encore pouvoir les restaurer plus tard.
- Migrer les instances d'un nœud Nova à l'autre en cas de montée en charge (ré-équilibrage dynamique).
- Se connecter à la console de chaque instance au travers d'un proxy VNC.

Si les principes de Nova semblent évident, l'implémentation est tout autre. La grande difficulté (qui est également une force du projet) réside cependant dans son algorithmie d'instanciation et d'équilibrage. En effet, lorsqu'un utilisateur requiert l'instanciation d'une nouvelle VM (différents modèles existent, plus ou moins bien dimensionnés), la demande est faite à Nova par l'API, puis retransmise aux différents nœuds de processing Nova existant via RabbitMQ. Chacun de ces nœuds héberge probablement déjà un certain nombre de VMs. Certaines seront certainement détruites prochainement, d'autres non. Chaque nœud dispose cependant de capacités physiques différentes (CPU, disque, mémoire) car les serveurs sont différents, mais dispose également d'une charge opérationnelle différente. C'est alors à Nova d'élire le nœud qui sera (à l'instant T du moins) le plus à même d'héberger la VM demandée. Plus facile à dire qu'à faire. D'autant qu'au cours du temps, certains nœuds seront amenés à être éteints (défaillance matérielle) et d'autres seront ajoutés (achat de nouveaux serveurs). L'orchestrateur interne de Nova doit donc prendre tout cela en compte en temps-réel afin de procéder au meilleur choix. Au cours du temps, il sera également de son ressort de faire migrer les VMs (de manière transparente pour l'utilisateur final) afin de mieux lisser la charge présente sur chaque nœud.

Notons cependant que Nova est un projet mature. Son implémentation provient du projet Nebula (cf. **[8]**) de la NASA et les sociétés HP et RackSpace fournissent des services commerciaux articulés autour de Nova, preuve de sa grande qualité.

5 Cinder Block Storage

Comme vu préalablement, Cinder est un composant interne à OpenStack (majoritairement utilisé par Nova), dont le rôle est de fournir des périphériques de stockage en mode bloc (i.e. des disques durs) aux instances virtuelles des nœuds de computing. L'API de Cinder est relativement simple et propose les fonctionnalités suivantes :

- Création, modification et destruction de volumes disques.
- Photo instantanée (« snapshot ») et sauvegarde de volumes (vers le composant Swift)
- Requêtes de statut et récupération des métadonnées.

Les principaux consommateurs de l'API de Cinder sont bien évidemment Nova, mais également le module Horizon (le fameux « dashboard ») qui permet à l'utilisateur de gérer les volumes au sein d'une interface graphique (et de les assigner à une VM). Il n'y a évidemment pas de limite de nombre de volumes associables à une VM donnée.

L'architecture de Cinder propose en outre un certain nombre de backends vers différentes plateformes ou protocoles de stockage de données, comme représentée au sein de la figure 4.

Cinder peut bien entendu utiliser un stockage disque local au serveur Linux l'hébergeant, mais également s'interfacer vers des solutions de stockage de type SAN, propriétaires et extrêmement coûteuses, telles que les solutions Storwize SVC d'IBM, NetApp, SolidFire, Nexenta StorSAN, Zadara ou encore LeftHand SAN d'HP, pour ne citer qu'elles. Il existe d'autres alternatives (que vous pourrez mettre en place vous même), plus économiques, basées sur des solutions de systèmes de fichiers réseaux distribués telles que GlusterFS (cf. **[9]**) de Red Hat, un cluster Ceph RADOS (cf. **[10]**) ou encore un cluster Sheepdog (cf. **[11]**). Si votre vocation n'est pas nécessairement de monter un data-center gérant des péta-octets ou exa-octets, vous pourrez également vous tourner vers des solutions de type iSCSI (cf. **[12]**, à noter que les différents NAS semi haut de gamme du marché grand public supportent ce protocole).

Comme c'était déjà le cas pour Nova, Cinder peut être déployé sur plusieurs nœuds, chacun pouvant avoir des backends différents. Les requêtes d'écriture envoyées à Cinder sont alors prises en

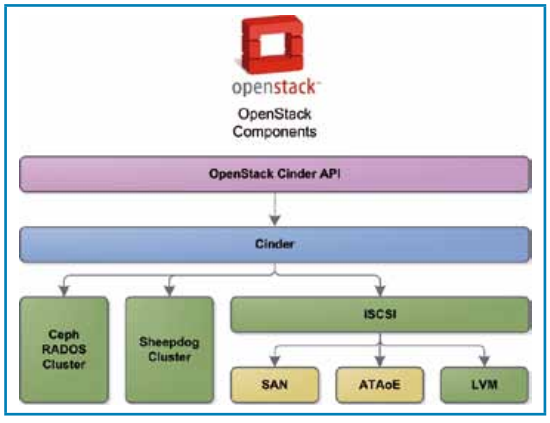


Figure 4 : Architecture du composant Cinder Block Storage

compte par l'ordonnanceur interne à Cinder qui décidera vers quel nœud les rediriger. L'enjeu de cet ordonnanceur (et de son algorithme) étant de trouver le fournisseur de stockage par bloc optimal pour le volume désiré. En effet, une VM hébergeant une base de données fortement sollicitée nécessitera un disque plus rapide que les autres (que l'on mesure en IOPS ou « I/O par seconde », le nombre de requêtes d'écriture/lecture possibles par seconde). Il n'est donc pas rare de trouver des combinaisons mixtes entre disques durs traditionnels et disques SSD ultra-rapides. Le client, lors de la location d'un volume Cinder peut alors (s'il en met le prix), demander à avoir un volume rapide à fort IOPS, avec un débit soutenu garanti.

Notez enfin que si l'installation dite « basique » de Cinder propose d'utiliser votre disque local (via iSCSI au dessus de LVM), car très simple en terme de déploiement et configuration, cela ne conviendra que pour des besoins de tests. En effet, je ne saurai que trop vous recommander de déployer ne serait ce qu'un vrai système iSCSI distant (au travers d'un NAS par exemple). Le premier exemple convient si vous déployez toute la suite OpenStack sur la même (grosse) machine ou si vous utilisez la même machine pour Nova et Cinder. Mais si vous disposez de plusieurs nœuds Nova et souhaitez pouvoir migrer les VMs d'un nœud à l'autre, il sera nécessaire d'utiliser des volumes disques distants, sous peine de devoir recopier les volumes Cinder d'une machine à l'autre à chaque migration de VM (opération longue et coûteuse).

6 Quantum Networking

Quantum (également appelé Neutron désormais, depuis Juin 2013) propose un service de NaaS, ou « Network as a Service ». Comme pour les disques, CPU et autres mémoires, le réseau est vu comme une ressource et le but de Quantum est d'assurer la gestion de ce ou ces réseau(x) ainsi que des adresses IP associées. En pratique, l'API de Quantum permet aux utilisateurs de créer leurs propres sous-réseaux (supposons que vous louiez plusieurs VMs chez un hébergeur et souhaitiez que ces différentes machines appartiennent au même sous-réseau d'entreprise) et d'y attacher des interfaces virtuelles. Certaines de vos VMs seront utilisées en interne, d'autres afficheront des services frontaux et devront être accessibles depuis Internet (ou l'extérieur d'une manière générale). Quantum vous permettra alors également d'associer une (ou plusieurs) adresse(s) IP publique(s) aux machines de votre réseau privé, comme le présente la figure 5.

Quantum propose différents modèles réseaux, pour répondre aux besoins spécifiques à chacune de vos applications. Libre à vous donc de créer des réseaux dits « à plat » ou d'utiliser des VLAN pour séparer votre trafic. L'adressage IP peut se faire de manière statique ou par DHCP et il est possible de

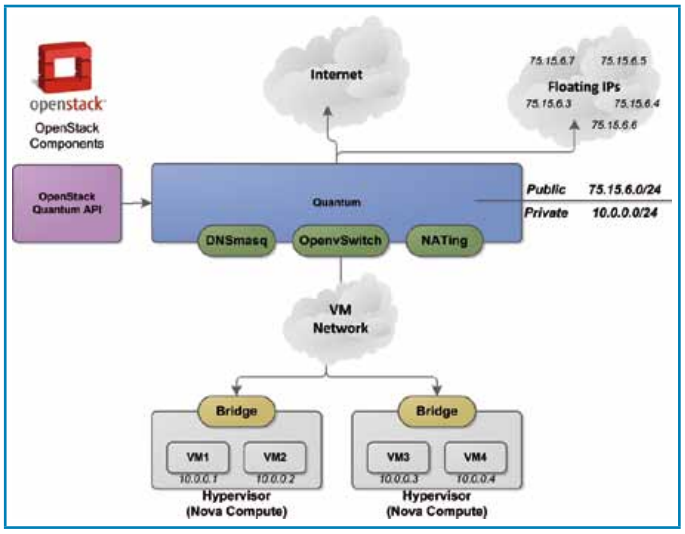


Figure 5 : Architecture du composant Quantum (ou Neutron) Networking

louer un ensemble d'adresse IP (notion de « Floating IPs ») que vous pourrez dynamiquement associer et router vers différentes instances. En cas de maintenance de services, cela permet par exemple de basculer d'un serveur à l'autre de manière transparente pour l'utilisateur du service.

Comme à l'accoutumée avec OpenStack, Quantum adopte une architecture modulaire permettant l'intégration avec différentes solutions réseaux propriétaires ou Open Source et dispose d'un framework d'extension autorisant l'ajout de nouveaux services tels qu'un détecteur d'intrusions (ou IDS, pour « Intrusion Detection System »), un répartiteur de charge (« load balancer », un pare-feu ou encore des réseaux privés virtuels (VPN). Dans sa dernière version, Quantum supporte la connexion avec les solutions propriétaires de switching (physiques et virtuels) de Cisco, les produits OpenFlow de NEC et NVP de Nicira ou encore le système d'exploitation réseau de Ryu. Du côté des extensions vers des solutions libres, la solution de déploiement la plus simple et rudimentaire va vers le projet Linux Bridging (cf. **[13]**) et la solution la plus complexe mais aussi la plus évoluée se retrouve dans le projet Open vSwitch (cf. **[14]**). Notez enfin que Quantum supporte le protocole OpenFlow (cf. **[15]**), tel que défini par l'ONF (« Open Networking Foundation », cf. **[16]**), adoptant la philosophie dite SDN (pour « Software Defined Network », cf. **[17]**), permettant aux réseaux d'être intelligents et de s'auto-configurer, mais également de définir les chemins réseaux à suivre par vos paquets IP.

7 Horizon Dashboard

Si les APIs REST d'OpenStack ou l'API de compatibilité avec Amazon permettent aux développeurs de gérer et d'interconnecter les différentes ressources du projet, il fallait

bien également penser à l'utilisateur final. Qu'il soit administrateur ou simple client, ce dernier a la possibilité, au travers du composant Horizon, d'accéder aux différentes ressources (de les louer, assigner, configurer ...) via une interface graphique Web (reposant sur le framework Django), telle que représentée au sein de la figure 6.

Parmi les principales fonctionnalités du composant Horizon, nous retrouvons donc la possibilité (pour l'utilisateur final) de :

- Récupérer les informations d'usage et les quotas.
- Contrôler (démarrer, éteindre, mettre en pause ...) vos différentes instances de VM.
- Contrôler (créer, détruire, associer ...) vos volumes de stockage disque.
- Créer des photos instantanées (« snapshots ») de vos instances de VM.
- Gérer le contrôle d'accès et les règles de pare-feu de vos instances de VM.
- Visualiser graphiquement votre topologie réseau (au travers de Quantum).

D'un point de vue administrateur, Horizon propose quelques fonctionnalités supplémentaires, parmi lesquelles la possibilité de :

- Rajouter des offres au catalogue client (nouvelle VM « type », disposant de X cœurs CPU, Y Go de disque et Z Go de mémoire).
- Gérer les comptes utilisateurs et la notion de groupes logiques.
- Visualiser le statut des différents services opérant dans votre Cloud ainsi que les quotas et limites associées à chaque projet.
- Envoi de modèles d'images de VM vers Glance.
- Migration à la volée d'instances d'un nœud de computing Nova à un autre.

8 Swift Object Storage

Et terminons enfin par le plus complexe mais également le principal (avec Nova) composant d'OpenStack, à savoir le projet Swift qui permet la gestion et le stockage de données sous forme d'objets. Nova et Swift furent les deux premiers composants d'OpenStack. Le premier a été conçu par la NASA (via le projet Nebula) , le second par la société d'hébergement RackSpace (via son projet Cloud Files, cf. **[18]**). Si Nova est une surcouche à un ensemble de technologies de virtualisations existantes, Swift a le mérite d'être une solution dédiée et construite de zéro. Swift se veut un peu différent des

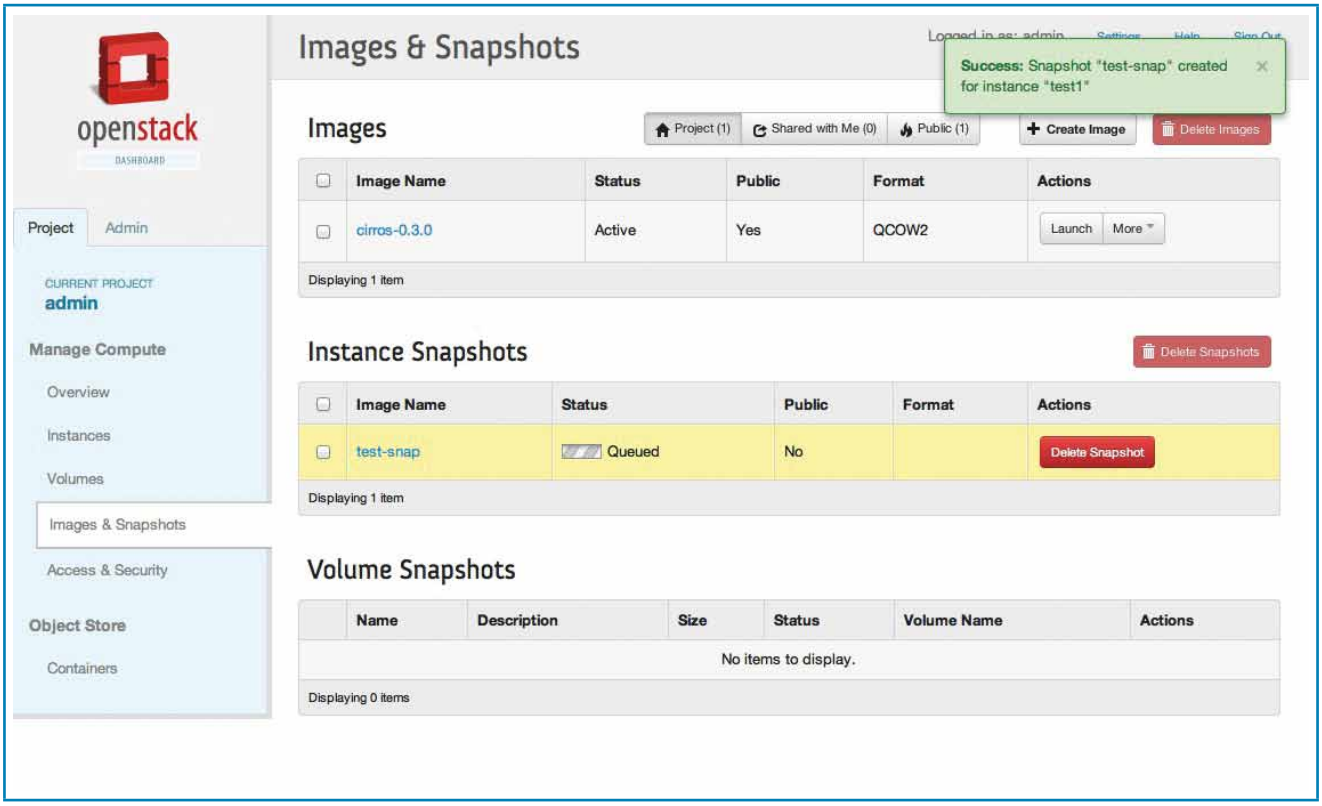


Figure 6 : Interface graphique Web du composant Horizon Dashboard

autres composants d'OpenStack car quelque peu indépendant. Un Cloud OpenStack peut en effet se déployer sans Swift (si seule la virtualisation vous intéresse). De même, il n'est pas rare de retrouver des sociétés d'hébergement proposant des offres de stockage de données (e.g. SwiftStack, cf. **[19]**) , qui repose sur Swift en interne, bien que ne disposant d'aucune fonctionnalité de virtualisation.

Pour faire simple et enlever tout de suite toute idée préconçue : Swift n'est PAS un système de fichiers. Swift permet de stocker et récupérer des fichiers au travers d'une API REST, qui sont stockées sous la forme d'objets dans une grande base de données ou table de hachage géante mais ça s'arrête là. Swift est conçu pour stocker des données résiduelles à long terme et non pas des fichiers qui sont continuellement accédés ou modifiés. Il n'y a aucune notion d'inode ou de parenté entre les objets comme on pourrait en retrouver dans un système de fichiers (local ou même distribué). A chaque objet peuvent cependant être associées des métadonnées (qui au final permettent de créer un simili système de fichiers si on le désire vraiment, mais ce n'est pas le but premier).

Swift a été conçu pour être très largement scalable et pouvoir assurer une croissance toujours plus forte des données (on parle d'une consommation mondiale de données en 2020 qui serait 50 fois supérieure à ce qu'elle n'a été en 2010, cf. **[20]**). Il est donc nécessaire de pouvoir assurer cette montée en volume, tout en garantissant l'intégrité des données stockées et la rapidité d'accès à l'information. Les objets gérés par Swift sont ainsi écrits sur de multiples disques, disséminés autour d'un grand nombre de serveurs au sein d'un (ou de plusieurs) data-center(s). Swift est alors responsable de garantir la synchronisation et réplication des données et de garantir leur intégrité au sein du cluster. La force de Swift réside dans sa capacité à croître horizontalement, « simplement » en ajoutant de nouveaux nœuds (serveurs et disques) au système, le tout sans interruption du système global. Si un disque venait à tomber en panne, il suffit de le remplacer, ses données ayant déjà été répliquées au sein d'autres composants du cluster. La force de Swift réside également dans son approche 100% logicielle. Aucun composant matériel spécifique n'est nécessaire et, disques comme serveurs, sont tout ce qu'il y a de plus « classiques ». Vous n'aurez donc pas besoin (il est d'ailleurs plus que recommandé de ne pas en avoir) de solutions RAID, qu'elles soient logicielles ou matérielles.

D'un point de vue de l'utilisateur final, Swift propose les fonctionnalités ci-dessous :

- Stockage et accès aux objets distants (fichiers) au travers d'une API REST.
- Association et modification de métadonnées associées à chaque objet.
- Gestion de version multiples par objet.
- Services de pages web statiques (i.e. HTML / JS) et d'objets par HTTP.

A l'usage, il est très simple de récupérer l'ensemble des objets stockés dans un conteneur donné (équivalent d'un répertoire dans un système de fichiers classique), via la requête :

```
GET http://swift.exemple.com/v1/account/container/
```

De même, stocker un objet se fait très facilement, via :

```
PUT http://swift.exemple.com/v1/account/container/mon_objet
```

Difficile de faire plus simple ! Chaque objet présent dans **Swift** dispose de sa propre URL d'accès. En outre, les données d'un objet peuvent être physiquement situées n'importe où dans le cluster (c'est transparent pour l'utilisateur des données) et chaque objet est répliqué trois fois par zone (groupe logique de disques).

Mais voyons maintenant plus en détail l'architecture interne de Swift, comme le présente la figure 7.

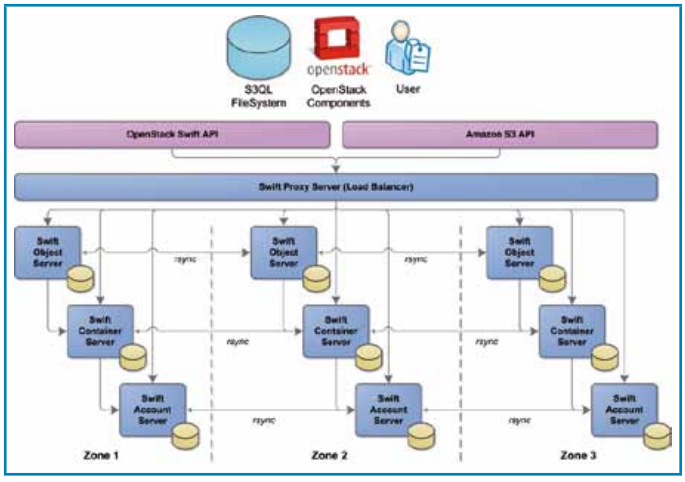


Figure 7 : Architecture du composant Swift Object Storage

Swift se découpe en plusieurs blocs logiques, parmi lesquels ont retrouve :

- Le « Proxy Server » qui constitue la face frontale de Swift et qui intercepte les requêtes en provenance de l'API (celle de Swift ou celle compatible avec Amazon S3). Le Proxy Server agit comme un répartiteur de charge, redirigeant la requête vers le nom concerné, en fonction de l'URL demandée. Dans un scénario de production, il y a souvent un minimum de 2 serveurs proxy (avec redirection DNS par Round-Robin) afin de garantir la haute disponibilité s'il l'un des serveurs venait à avoir un problème. Certains Proxy Server utilisent également MemCached (cf. **[21]**) de manière à accroître encore davantage les performances, vers les objets ou requêtes fréquemment appelées.

- Les « Zones » qui correspondent à des répartitions logiques (mais qui peuvent également être physiques) des serveurs et disques et permettent de créer des groupes de réplication, garantissant un accès constant aux données. Les données sont répliquées entre zones. Ainsi, si une zone venait à disparaître, ou ne plus être disponible pendant un moment, le système garantit toujours l'intégrité des données. La plus petite zone possible correspond à un disque dur. Il est bien sûr (et c'est évidemment recommandé), de l'élargir à un ou plusieurs racks de serveurs, potentiellement dans des emplacements différents de votre data-center ou mieux, dans des data-centers géographiquement distincts. Par défaut, Swift assure une triple réplication des données (c'est bien entendu configurable). L'algorithme de réplication, lors de la duplication d'une donnée nouvellement écrite, tendra a préférer un serveur dans une zone non utilisée, qu'un serveur non utilisé dans une zone qui contient déjà un dupliqua des données.
- Les « Account » et « Container » (comptes et conteneurs) sont des bases de données indépendantes, bien que distribuées au sein du cluster. Un compte contient une liste de conteneurs (l'équivalent d'un répertoire ou dossier). Un conteneur, pour sa part, contient une liste d'objets.
- Les « Objects » correspondent aux données brutes (les fichiers) qui sont stockées et auxquelles on associe le plus souvent des métadonnées. En règle générale, les serveurs d'objets utilisent XFS comme système de fichiers pour y stocker les différentes données.

Enfin, et c'est là toute la force de Swift, le système dispose d'une entité de réplication des données qui assure la consistance du système. Il s'agit d'un processus périodique qui maintient une copie parfaite des données entre les différentes zones. La comparaison entre zones se fait par différentiation sur le hachage MD5 des objets (ou conteneurs) présents dans chaque zone et c'est la zone qui contient la version la plus récente de l'objet qui devient maître. La réplication se fait ensuite par des mécanismes simples basés sur rsync.

9 Supervision

Nous l'avons vu, la solution OpenStack se veut très complète et permet de créer et déployer son propre Cloud pour concurrencer les plus grands. Oui mais voilà, au fur et à mesure que votre topologie réseau et système évolue, sa maintenance se complexifie. Il est grand temps de remettre de l'ordre dans tout cela et donc de superviser l'ensemble. Et c'est bien là le plus gros problème. Si vous avez bien suivi (cf. tableau 2), le composant de supervision Ceilometer n'en est encore qu'à l'état d'incubateur et son arrivée officielle au sein d'OpenStack ne devrait pas voir le jour avant la fin de

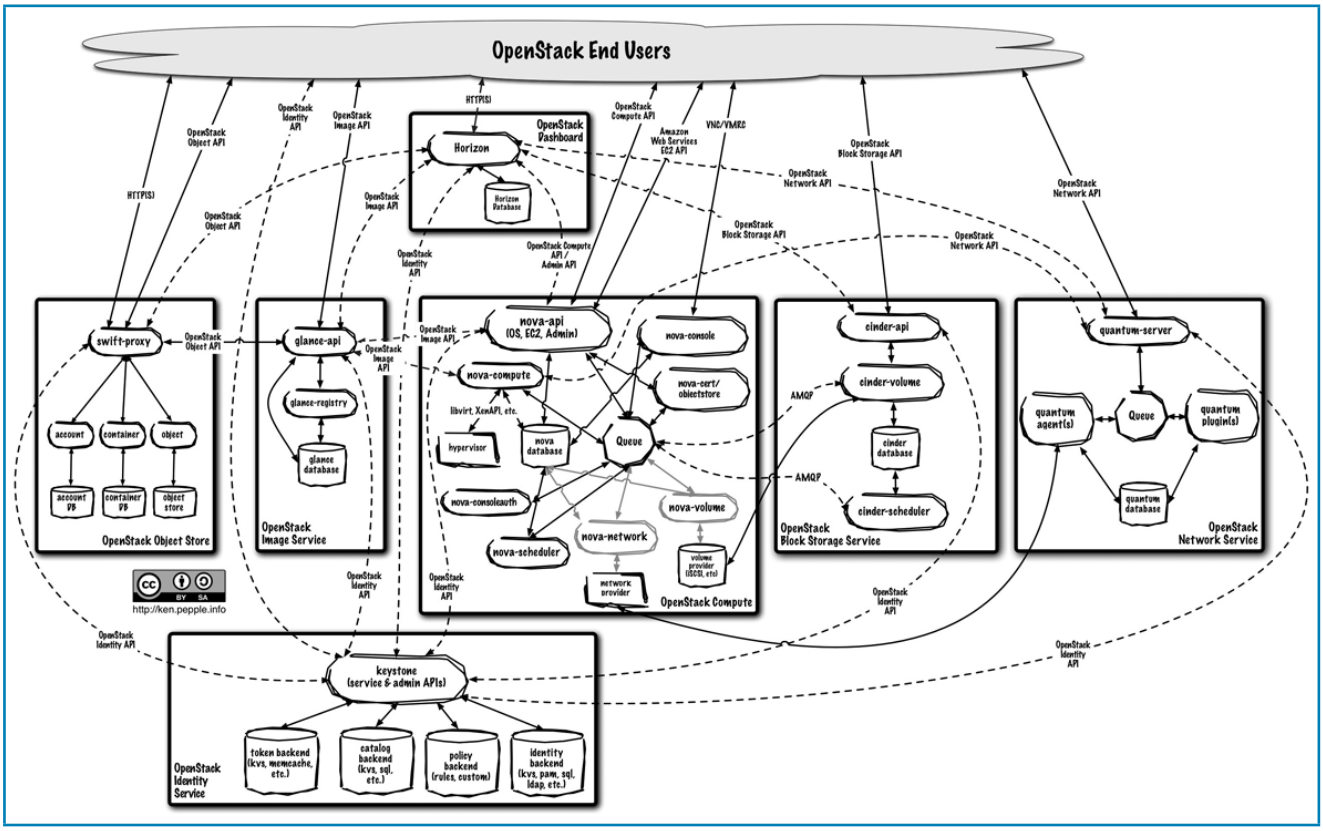


Figure 9 : Vue logique architecturale d'OpenStack (par Ken Pepple)

l'année. Les différentes APIs d'OpenStack (principalement celles de Nova) et les interfaces en ligne de commande (CLI) à votre disposition pourront aider mais ne sont guère suffisantes. En attendant Ceilometer, vous pourrez donc vous en remettre à différents projets Open Source tels que Monit (cf. **[22]**), Nagios (cf. **[23]**), Collectd (cf. **[24]**) et Graphite (cf. **[25]**) pour vous permettre de superviser ce qui se passe tant au niveau applicatif, qu'au niveau de la machine virtuelle ou de l'ensemble des nœuds de computing Nova qui constituent votre réseau, tel que présenté au sein de la figure 8.

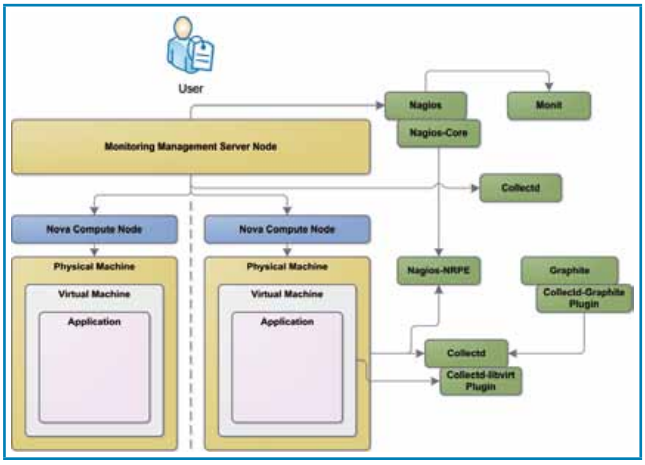


Figure 8 : Supervision des nœuds de computing Nova

10 Pour aller plus loin ...

Le projet OpenStack, bien qu'encore très jeune, avance à grands pas et a su fédérer de nombreuses grandes entreprises afin de bâtir une solution libre et Open Source d'infrastructure Cloud de type IaaS digne de concurrencer les plus grandes solutions propriétaires, dont celle d'Amazon. En imposant une API devenue référence et en assurant la compatibilité avec les APIs EC2 et S3 d'Amazon, OpenStack permet une migration en douceur de vos applications du Cloud public Amazon vers votre propre Cloud, privé ou public. Qu'il s'agisse de basculer complètement ou simplement pour assurer une solution complémentaire de haute-disponibilité de vos services, OpenStack s'impose comme la référence dans le domaine. Son utilisation de technologies libres et matures (la virtualisation par KVM en tête ou encore la gestion réseau par Open vSwitch) et son adoption par les grandes distributions GNU/Linux orientées serveur qu'offrent Red Hat et Ubuntu en font un acteur incontournable du marché.

Mais le projet reste encore jeune et souffre néanmoins de certaines lacunes. Son orientation multi-composants est extrêmement séduisante (et facilite grandement la tâche des développeurs) mais peut également présenter un frein à quiconque cherche à déployer l'ensemble. Je vous laisse prendre conscience de l'ampleur des inter-dépendances au sein de l'excellent schéma d'architecture produit par Ken Pepple en figure 9.

Conclusion

Déployer OpenStack n'est pas insurmontable, loin de là, mais peut quand même constituer un challenge pour tout SysAdmin ou DevOps qui se respecte. La complexité de son déploiement est d'ailleurs souvent le principal reproche adressé au projet. Pire, la mise à jour vers une nouvelle version (tous les 6 mois) se passe généralement très mal. Et une fois déployé, l'absence (pour l'instant) de véritable solution intégrée de supervision, de mesure, d'analyse, de planification de ressources, de log, d'orchestration ou encore de facturation laisse un goût quelque peu amer. Pour autant, le jeu en vaut la chandelle, à condition d'y mettre les moyens (financiers et humains). C'est ainsi que nous verrons, au cours du prochain numéro, comment déployer notre propre installation d'OpenStack et y déployer notre solution collaborative à la Google Drive, reposant sur le projet OwnCloud (cf. **[26]**) au travers d'une architecture pensée Cloud. A suivre ! ■

Références

[1] http://aws.amazon.com/fr/
[2] http://api.openstack.org/
[3] http://aws.amazon.com/fr/s3/
[4] http://www.amqp.org/
[5] http://www.rabbitmq.com/
[6] http://libvirt.org/
[7] http://aws.amazon.com/fr/ec2/
[8] http://nebula.nasa.gov/
[9] http://www.gluster.org/
[10] http://ceph.com/category/rados/
[11] http://www.osrg.net/sheepdog/
[12] http://fr.wikipedia.org/wiki/Iscsi
[13] http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge
[14] http://openvswitch.org/
[15] http://www.openflow.org/
[16] https://www.opennetworking.org/
[17] http://fr.wikipedia.org/wiki/Software_Defined_Networking
[18] http://www.rackspace.com/cloud/files/
[19] http://swiftstack.com/
[20] http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf
[21] http://memcached.org/
[22] http://mmonit.com/monit/
[23] http://www.nagios.org/
[24] http://collectd.org/
[25] http://graphite.wikidot.com/
[26] http://owncloud.org/

ANDROID : PLUGIN DE LAYOUT

par Philippe PRADOS [Consultant OCTO Technology]

Est-il possible d'ajouter des extensions à une application Android, et particulièrement des extensions à un layout ? C'est ce que nous allons voir.

Les entreprises utilisent de plus en plus des terminaux Androids pour offrir des applications à leurs employés. C'est particulièrement le cas des entreprises de coursiers ou de relevés de compteurs. En effet, les agents parcourent des kilomètres dans des environnements difficiles au niveau connexion, pour offrir de plus en plus de services innovants. Tous les intervenants itinérants ont vocation à utiliser les technologies mobiles, mais malheureusement, dans des conditions au niveau réseau, pas toujours idéales.

Les applications de gestion des tournées doivent offrir de plus en plus de fonctionnalités. Certaines sont spécifiques à une localisation géographique, d'autres sont partiellement en tests ou dépendant des profils des employés.

Une application Android monolithique n'est plus la solution. Il faut concevoir une application pouvant accueillir des extensions. Cela permet, par le simple choix des APK installés, d'enrichir les tournées des agents. Un *Mobile Device Managment* (MDM) permet alors d'installer le jeu d'extension correspondant à chaque profil. Lors de la mise à jour d'un composant, seul ce dernier est impacté.

De nombreux scénarios peuvent bénéficier de cela. Avant toute chose, les sources sont présent ici : https://github.com/pprados/android-pluginviews.

1 Communication avec les plugins

La première étape à franchir est de choisir une technologie pour la communication avec les plugins. Ajouter un plugin, c'est ajouter un APK. Donc nous devons permettre à deux APK de communiquer.

Android nous propose plusieurs approches, mais la plus adapté est d'exposer un composant aux autres applications via une interface. Cela s'effectue très simplement en quelques étapes.

La première chose à faire est de décrire l'interface de communication. On ne va pas utiliser Java pour cela, mais une syntaxe très proche : l'AIDL. Un fichier avec cette extension est traité par l'environnement de développement d'Android pour générer du code servant pour le client et pour le serveur.

Un fichier AIDL est composé d'une première instruction pour indiquer le nom du package où devrons être généré les classes. Ensuite, on peut trouver des imports pour récupérer les structures des autres AIDL.

Enfin, une interface décrit les différentes méthodes avec les paramètres et leurs types.

```
package fr.prados.pluginview.provider;
import android.os.Bundle;

interface RemoteViewProvider
{
  int getSize(in Bundle bundle);
}
```

Pour les paramètres de type **`Object`**, il faut indiquer si ce dernier sera utilisé uniquement en lecture par la méthode appelée (attribut **`in`**), sera utilisé uniquement pour que la méthode appelée y place des informations (attribut **`out`**) ou une combinaison des deux (attribut **`inout`**).

Les objets sont envoyés par « valeur », après avoir été sérialisés via les **`Parcel`** (sérialisation spécifique d'Android, bien plus efficace que la sérialisation java classique). Ils doivent alors avoir leurs propres fichiers AIDL pour déclarer cela.

```
package android.os.Bundle ;
parcelable Bundle;
```

Pratiquement toutes les structures de données d'Android possèdent leurs fichiers AIDL dans les sources. Certains sont déjà connus du compilateur AIDL. C'est le cas pour **`Bundle`**. Nous pouvons alors simplifier notre déclaration en supprimant l'import et le fichier **`android/os/Bundle.aidl`**.

```
package fr.prados.pluginview.provider;

interface RemoteViewProvider
{
  int getSize(in Bundle bundle);
}
```

Un paramètre peut être une structure exposée par un autre service. C'est comme cela que l'on peut monter des call-backs entre APKs.

Avec un fichier comme celui-là dans votre projet, le compilateur va générer une classe **RemoteViewProvider** avec une interface **RemoveViewProvider.RemoveViewProvider** et deux sous-classes **RemoveViewProvider.Stub** et **RemoteViewProvider.Stub.Proxy**. Cette dernière ne nous intéresse pas, sauf à creuser le protocole. C'est l'objet d'autres articles que j'ai publiés.

La deuxième chose à faire est de proposer une implémentation du service. C'est super simple à faire. Il suffit de créer une classe qui hérite de **RemoteViewProvider.Stub**.

```
public class RemoteViewProviderImpl extends fr.prados.pluginview.
provider.RemoteViewProvider.Stub
{
  ...
}
```

On a déclaré notre interface et une implémentation. Il faut maintenant permettre aux autres applications de se connecter à une instance **RemoteViewProviderImpl**.

Dans les autres architectures d'IPC, il faut généralement s'adresser à un annuaire. Sous Android, une utilisation particulière des services permet d'adresser cela.

Nous devons rédiger un service simple, surchargeant la méthode **onBind()** pour livrer une instance de **RemoteViewProviderImpl**.

```
package fr.prados.pluginview.provider1;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class ProviderService extends Service
{

  @Override
  public IBinder onBind(Intent intent)
  {
    return new RemoteViewProviderImpl(this);
  }
}
```

Une dernière étape, déclarer le service dans le fichier **AndroidManifest.xml**.

```
<service
  android:name=".ProviderService"
  android:permission="fr.prados.PLUGINVIEW"
  android:exported="false"
>
  <intent-filter>
    <action android:name="fr.prados.pluginview.providers" />
    <category android:name="android.intent.category.DEFAULT" />
  </intent-filter>
</service>
```

Et voilà. Avec deux classes, un fichier AIDL et la déclaration du service dans le **manifest**, nous sommes capable d'invoquer un traitement présent dans un autre APK.

Mais, c'est en fait plus compliqué que cela. En effet, le code généré à partir du fichier AIDL doit être utilisé à la fois par le client du service et par le producteur du service. Il faut réorganiser ces fichiers dans différents projets Android.

La recette n'est pas très compliquée. Créez trois projets Android. L'un pour le client, l'un pour le serveur, et le dernier, un projet Android Library avec uniquement le fichier aidl placé dans le bon répertoire du package.

Les deux premiers projets référencent le projet librairie avec l'AIDL. Ainsi, le code généré est disponible pour les deux parties.

Dans le projet serveur, on place la classe du service et la classe d'implémentation et on déclare le service dans le **manifest**.

Ce qui est amusant, c'est que l'on peut faire plusieurs projets d'implémentations du service. Le client pourra alors tous les retrouver pour gérer une collection de plugins.

2 Découverte des plugins

Ok, nous avons un projet client qui souhaite découvrir tous les plugins. Pour cela, il faut utiliser :

```
mContext.getPackageManager().queryIntentServices(mIntent, 0);
```

Il faut utiliser un **Intent** construit avec l'**action** déclarée dans le manifest.

```
mIntent=new Intent("fr.prados.pluginview.providers")
```

Cela permet de récupérer la liste de toutes les applications implémentant un service répondant à l'**Intent**.

Nous pouvons alors itérer sur les **ServiceInfos** pour demander ensuite une connexion directe avec chacun.

```
for (ResolveInfo info : providersInfo)
{
  final ServiceInfo serviceInfo = info.serviceInfo;
  if (serviceInfo == null)
    continue;
  final Intent intent=new Intent(mIntent);
  // Select specific package
  intent.setClassName(serviceInfo.packageName, serviceInfo.name);
     ...
  boolean rc = mContext.bindService(
    intent, conn, Context.BIND_AUTO_CREATE);
}
```

J'ai fait abstraction de l'instance **ServiceConnection** qui propose deux call-backs. L'une est invoquée lorsque le service a réussi à se connecter, l'autre lorsque le service est déconnecté (perte du processus de l'APK du plugin par exemple).

Pour déconnecter un service, il faut réutiliser la même instance de **ServiceConnection** pour faire un **unbindService()**.

Pour gérer correctement un ensemble de plugins, il faut donc garder la liste des **ServiceConnection** en parallèle de la liste des instances *remotes*.

Lorsqu'une connexion avec un plugin est effective, la méthode **onServiceConnected()** de la callback reçoit un paramètre **IBinder** qu'il suffit de convertir pour répondre à l'interface. En fait, c'est un *cast* particulier via une méthode générée.

```
@Override
public void onServiceConnected(ComponentName className, IBinder service)
{
  RemoteViewProvider provider=RemoteViewProvider.Stub.asInterface(service)
  ...
}
```

Cette méthode vérifie si l'instance référencée par **service** est une instance locale, dans ce cas un cast classique est suffisant. Si l'instance est distante, la méthode crée une instance **Proxy** compatible avec l'interface. Ainsi, tous les usages de l'interface seront propagés à une instance présente dans un autre APK.

3 Généralisons la gestion des plugins

Bon, nous savons comment proposer des plugins et comment s'y connecter. Si nous souhaitons faire une classe générique pour gérer automatiquement cela, c'est plus compliqué.

Nous souhaitons créer une classe qui gère un type générique généré par un AIDL.

```
public class BindProviders<T extends android.os.IInterface>
{
  ...
}
```

Cette classe va recevoir l'**Intent** en paramètre pour découvrir les plugins, se connecter sur chacun et gérer le cycle de vie des connexions. Une call-back sera invoquée lorsque tous les plugins seront connectés.

```
/**
 * A callback called when all providers are connected.
 * @author Philippe Prados
 *
 * @param <T> AIDL interface
 */
public interface CallBack<T>
{
  /**
   * Call-back invoked when all providers are connected.
   * @param providers
   */
  void onProvidersBinded(List<T> providers);
}
```

Comme nous devons garder l'instance **ServiceConnection** et l'instance du proxy du plugin connecté, je déclare une sous-classe comme ceci.

```
interface ProviderServiceConnection<T> extends ServiceConnection
{
  T getProvider();
}
```

Je peux alors garder deux listes.

```
// Providers and connections
private List<ProviderServiceConnection<T>> mProvidersConnections=
  new ArrayList<ProviderServiceConnection<T>>();
// Only providers
private List<T> mProviders=new ArrayList<T>();
```

Pour le moment, tout va bien. La difficulté arrive lorsque l'on souhaite convertir le **IBinder** lors d'une méthode **onServiceConnected()** en **T**. On souhaite appeler :

```
mProvider = (T)T.Stub.asInterface(service)
```

Mais ce n'est pas possible. Le mécanisme de généricité en Java perd le type **T** à l'exécution (*type Erasure*). Nous devons alors procéder autrement.

```
Class<?> clazz=Class.forName(mParameterType.getName()+"$Stub");
Method asInterface=clazz.getMethod("asInterface", android.os.IBinder.class);
mProvider = (T)asInterface.invoke(null, service);
```

Cela nous impose de posséder le type de **T** à l'exécution. Le constructeur le demande car il ne peut le découvrir.

```
public BindProviders(
  Context context,
  Intent intent,
  Class<T> parameterType,
  CallBack<T> callback)
{
  mContext=context;
  mIntent=intent;
  mParameterType=parameterType;
  mCallback=callback;
}
```

Maintenant, c'est bon. Nous pouvons proposer une méthode **open()** pour connecter tous les plugins et une méthode **close()** pour s'y déconnecter.

La méthode **getProviders()** retourne la liste des providers connectés.

Pour utiliser cette classe avec l'exemple d'AIDL ci-dessus, il faut simplement rédiger

```
new BindProviders<RemoteViewProvider>(context,intent,
RemoteViewProvider.class,this)
```

Dans un fragment, il faut invoquer la méthode **open()** dans **onAttach()** et la méthode **close()** dans **onDetach()**.

```
@Override
public void onAttach(Activity activity)
{
  super.onAttach(activity);
  // Connect to all providers
  mProviderBinder=new BindProviders<T>(activity,mIntent,mParameterType,
this);
  mProviderBinder.open();
}
@Override
public void onDetach()
{
  super.onDetach();
```

GREAT PLACE TO WORK® Best Workplaces 2013 France

UNIVERSUM TOP 100 IDEAL EMPLOYER 2013 STUDENT SURVEY

IMAGINER ET CONCEVOIR UN MONDE SANS CONTRAINTE

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

LES TECHNOLOGIES JAVA, ANDROID ET LINUX N'ONT PAS DE SECRETS POUR VOUS ?
BOUSCULEZ TOUS LES CODES SUR **WWW.RECRUTE.PARROT.COM**

UN MONDE D'INNOVATIONS AU CŒUR DE PARIS

Parrot®

anthemoon

```
  // Deconnect all providers
  mProviderBinder.close();
  mProviderBinder=null;
}
```

Pour éviter les fuites mémoires, il est préférable de s'appuyer sur le cycle de vie des composants graphiques pour gérer les plugins. Vouloir faire le malin en gardant la liste en cache est rarement une bonne idée.

4 Enrichissement d'un Layout

Fig. 1

Parfait, nous avons une callback qui nous donne la liste des plugins. Mais pour le moment, nous ne pouvons qu'invoquer des traitements distants. Comment enrichir notre interface utilisateur à l'aide de plugin ?

Notre objectif est de présenter une liste dont les lignes ont été alimenté par les plugins. Chacun pouvant produire plusieurs lignes (Fig. 1).

Pour pouvoir faire cela, nous allons utiliser la technique utilisée par Android pour gérer les notifications et les Widgets du bureau. Ces derniers sont des bouts d'applications proposés à l'utilisateur, alors que le processus correspondant n'est pas présent en tâche de fond.

En fait, les notifications et les widgets sont des layouts simplifiés avec une interaction avec l'utilisateur très réduite, généralement limitée à l'appui d'un composant ou au parcours d'une liste pour les dernières versions. Le layout est récupéré par le framework en interrogeant l'application, puis gardé au chaud. L'application peut alors être tuée par le système. Le layout est toujours disponible.

Nous devons alors proposer une méthode à notre AIDL qui permet de retourner un **`Layout`** initialisé par valeur. Il existe une classe pour cela : **`RemoteView`**.

Nous ajoutons une méthode à notre AIDL.

```
package fr.prados.pluginview.provider;

interface RemoteViewProvider
{
  int getSize(in Bundle bundle);
  android.widget.RemoteViews getLoadingView(in Bundle bundle);
}
```

Il n'est pas besoin de créer un fichier **`android.widget.RemoteViews.aidl`**, car Android le connait déjà.

Coté implémentation, nous devons proposer un layout simple, pouvant être présent dans une ligne d'une **`ListView`**.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:id="@+id/provider1">
  <TextView
    android:id="@+id/text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginTop="16dp"
    android:textColor="@color/textview_textcolor"
    android:textAppearance="?android:attr/textAppearanceLarge"
  />
  <Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_alignParentTop="true"
    android:text="Button" />
</RelativeLayout>
```

Nous avons un simple texte et un bouton.

Dans la méthode d'implémentation de **`getLoadingView()`** exposé par l'AIDL, nous pouvons instancier cela :

```
final RemoteViews remoteView=
  new RemoteViews(mAppPackage,
  R.layout.provider1);
```

Le premier paramètre est le package Android de l'application du plugin. Pourquoi ce paramètre ? Nous allons le voir plus loin.

Comme notre plugin peut proposer plusieurs lignes, nous devons les identifier. Pour cela, nous proposons de recevoir cette information dans le **`Bundle`** reçu en paramètre de la méthode.

```
int i=1;
if (bundle!=null) i=bundle.getInt("id", 1);
```

Nous pouvons alors modifier le texte avec la méthode suivante de **`RemoteView`**.

```
remoteView.setCharSequence(R.id.text, "setText", "Provider1 #"+i);
```

C'est un peu étrange comme façon de modifier le layout, livrer le nom d'une méthode et ses paramètres. Pourquoi ne pas modifier directement le texte dans la vue ? Tout va devenir plus clair dans un instant.

Nous voulons également que lors de l'appui sur la ligne, un **`Intent`** spécifique vers un écran de notre plugin soit déclenché.

```
final Intent intent=new Intent(mContext,Provider1Activity.class);
intent.putExtra("id", i);
final PendingIntent pendingIntent=PendingIntent.getActivity(mContext,
  i, //request code must be different for each
  intent, 0);
remoteView.setOnClickPendingIntent(R.id.provider1, pendingIntent);
```

Attention, l'identifiant du **PendingIntent** doit être différent pour chaque ligne. Il est possible d'utiliser un simple numéro incrémenté à chaque fois. Nous avons choisi d'utiliser l'**id** de la ligne.

Si nous souhaitons avoir une réaction uniquement sur l'appui du bouton, la dernière ligne est différente.

```
remoteView.setOnClickPendingIntent(R.id.button1, pendingIntent);
```

Nous pouvons retourner notre instance **RemoteView**.

```
return remoteView ;
```

Il nous reste à résoudre la méthode **getSize()** pour indiquer combien de ligne le plugin génère.

```
@Override
public int getSize(Bundle bundle) throws RemoteException
{
  return 3; // arbitrary number of items
}
```

Nous voici avec la première implémentation d'un plugin. Chaque plugin peut proposer son layout.

5 Utilisation des layouts

Nous allons maintenant consommer nos layouts. Pour cela, nous créons un **ListFragment**.

```
public class RemoteListFragment
extends ListFragment
{
  ...
}
```

Nous injectons la tuyauterie comme décrit précédemment pour avoir une call-back invoquée avec la liste des plugins. Merci à **BindProviders<>**.

Il nous faut un **Adapter** pour faire le lien entre la liste et les plugins.

```
class ComplexRemoteViewAdapter extends BaseAdapter
{
  private Context mContext;
 private ArrayList<RemoteViewProvider> mItems=
    new ArrayList<RemoteViewProvider>();
  private Bundle mBundle=new Bundle();

  @Override
  public void notifyDataSetChanged()
  {
    super.notifyDataSetChanged();
    updateData();
  }
  public ComplexRemoteViewAdapter(Context context)
  {
    mContext=context;
    updateData();
  }
  @Override
  public int getCount()
  {
    return mItems.size();
  }

...
}
```

Avec une liste qui associe à chaque ligne, le **RemoteViewProvider**, on ajoute une petite méthode pour initialiser tous cela.

```
private void updateData()
{
  for (RemoteViewProvider provider:getProviders())
  {
    try
    {
      final int size = provider.getSize(mBundle);
      for (int i=0;i<size;++i)
        mItems.add(provider);
    }
    catch (RemoteException e)
    {
      // Ignore. Bug with the provider
    }
  }
}
```

La méthode la plus importante est **getView()**.

```
@Override
public View getView(int pos, View old, ViewGroup parent)
{
  try
  {
    final Bundle bundle=new Bundle();
    bundle.putInt("id", pos);
    final RemoteViews remoteViews =  mItems.get(pos).getLoadingView(bundle);
    final View view=remoteViews.apply(mContext,parent);
  }
  catch (RemoteException e)
  {
    e.printStackTrace();
    TextView tv=new TextView(parent.getContext());
    tv.setText("ERROR");
    return tv;
  }
}
```

La méthode **apply()** d'un **RemoteViews** permet de demander la construction de la vue décrite dans l'instance. La vue ainsi créée peut être retournée pour être affichée dans la **ListView**.

Première remarque, nous devons capturer l'exception **RemoteException** car nous invoquons un processus distant qui peut planter. Nous verrons comment faire en sorte que cette exception ne soit jamais lancée.

Dans le corps de la méthode, nous construisons un **Bundle** que nous alimentons avec le numéro de la ligne. Puis nous invoquons la méthode **getLoadingView()** du plugin.

Dans : la call-back du **Fragment**, nous rafraîchissons la liste.

```
@Override
public void onProvidersBinded(List<RemoteViewProvider> providers)
{
  if (mAdapter!=null)
    mAdapter.notifyDataSetChanged();
}
```

Pour la vue nous déclarons l'adapter :

```
@Override
public View onCreateView(LayoutInflater inflater,
  ViewGroup container,
  Bundle savedInstanceState)
{
  mAdapter=new ComplexRemoteViewAdapter(getActivity());
  setListAdapter(mAdapter);
  return super.onCreateView(inflater, container, savedInstanceState);
}
```

Cela fonctionne parfaitement. Nous avons bien un **ListFragment** dont chaque ligne est alimentée par un plugin. Lors de l'appui sur une ligne, un **startActivity()** est déclenché vers l'activité correspondante du plugin.

6 Gestion de la feuille de style

C'est bien, mais dans le layout du plugin, nous avons explicitement indiqué **textColor** et **textAppearance**.

```
<TextView
  ...
  android:textColor="@color/textview_textcolor"
  android:textAppearance="?android:attr/textAppearanceLarge"/>
```

Est-il possible d'utiliser la feuille de style de l'application racine pour afficher les layouts des plugins ? Cela serait cool !

En virant ces attributs, rien ne se passe comme prévu. Le texte est affiché en blanc sur fond gris clair. Intervenir sur la feuille de style du plugin ne fonctionne pas. Il faut nous plonger dans le code d'Android pour comprendre pourquoi.

On retrousse les manches, et c'est parti. En étudiant les sources de **RemoteView**, on est surpris de l'approche utilisée. À priori, on imagine qu'une vue est créée dans le plugin, puis sérialisée dans une **RemoteView** avant d'être envoyée à l'application racine et être reconstruite.

Comme toujours avec Android, le code n'est pas comme on l'imagine.

En fait, un **RemoteView** possède trois informations principales : le package de l'application qui porte le layout, le numéro du layout dans cette application (**R.layout...**) et une liste de commandes à appliquer pour modifier la vue créée.

Cela permet à la méthode **apply()** de créer un **Context** pour le package du plugin, pour lui demander de créer la hiérarchie de vues. Avec l'arbre en poche, la méthode applique successivement les traitements enregistrés par le plugin.

Par exemple, lorsque nous avons écrit ceci dans le plugin :

```
remoteView.setCharSequence(R.id.text, "setText", "Provider1 #"+i);
```

Cela n'a pas modifié la vue, mais ajouté une commande au **RemoteView** pour que cela soit appliqué localement par l'application racine. Il n'est donc pas possible d'utiliser un **RemoteView** sans layout associé.

Et comme le **LayoutInflater** est associé au **Context** de l'application du plugin, il ne peut utiliser la feuille de style locale.

Regardons plus précisément le code de la méthode **apply()** :

```
public View apply(Context context, ViewGroup parent, OnClickHandler handler)
{
  RemoteViews rvToApply = getRemoteViewsToApply(context);
  View result;
  Context c = prepareContext(context);
  LayoutInflater inflater = (LayoutInflater)
    c.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
  inflater = inflater.cloneInContext(c);
  inflater.setFilter(this);
  result = inflater.inflate(rvToApply.getLayoutId(), parent, false);
  rvToApply.performApply(result, parent, handler);
 return result;
}
```

Une première remarque. Le code ajoute un **Filter** à l'**inflater**. Cela va permettre d'invoquer la méthode **onLoadClass()** pour chaque vue du layout.

```
public boolean onLoadClass(Class clazz)
{
  return clazz.isAnnotationPresent(RemoteView.class);
}
```

En fait, cela permet de filtrer les vues compatibles avec les **RemoteViews**. Les classes correspondantes sont identifiées dans les sources d'Android avec l'annotation **@RemoteView**. Voilà une technique élégante pour gérer cela.

Ensuite, nous avons l'invocation de **prepareContext()** pour obtenir un **Context** spécifique. Regardons le détail de cette méthode.

```
private Context prepareContext(Context context)
{
  Context c;
  String packageName = mPackage;
  if (packageName != null)
  {
    try
    {
      c = context.createPackageContext(packageName, Context.CONTEXT_RESTRICTED);
    }
    catch (NameNotFoundException e)
```

```
    {
      Log.e(LOG_TAG, "Package name " + packageName + " not found");
      c = context;
    }
  }
  else
  {
    c = context;
  }
  return c;
}
```

Cool. C'est ce à quoi on s'attendait. Le code construit un **Context** sur l'application du plugin, pour pouvoir y récupérer les ressources.

Pour forcer la feuille de style, il faudrait pouvoir adapter ce **Context**, pour que la méthode **getTheme()** retourne notre thème. Nous devons alors manipuler à distance, la méthode **prepareContext()**.

Comme elle s'appuie sur le **Context** qu'on livre à la méthode **apply()**, essayons de modifier la méthode **createPackageContext()** pour wrapper la méthode **getTheme()**. La classe **ContextWrapper()** va nous aider.

```
final View view=remoteViews.apply(new ContextWrapper(mContext)
{
  public Context createPackageContext(String packageName, int flags)
    throws NameNotFoundException
  {
    Context context=
      new ContextWrapper(getBaseContext().createPackageContext(packageName,
flags))
      {
        // Delegate the theme to the context
        @Override
        public Theme getTheme()
        {
          return mContext.getTheme();
        }
      };
      return context;
  }
}, parent);
```

Super ! Cela fonctionne avec un Android 4.2 !

Jusqu'au jour ou je reçois la mise à jour 4.3 de l'OS. Vous savez, celle qui a fortement renforcée la sécurité. Et là, cela ne fonctionne plus. La raison est à trouver dans la nouvelle implémentation de la méthode **prepareContext()**.

```
c = context.createPackageContextAsUser(
  packageName, Context.CONTEXT_RESTRICTED, mUser);
```

Il n'est plus possible de gérer le thème car impossible de surcharger la méthode **createPackageContextAsUser()**.

Conclusion, ce n'est pas parce que les sources d'Android sont disponibles qu'il faut faire le malin :-)

Les layouts des plugins doivent indiquer tous les paramètres et ne pas s'appuyer sur la feuille de style.

7 Optimisation

Utiliser des plugins dans des applications différentes est une bonne idée, mais au niveau consommation de ressources mémoires, ce n'est pas l'idéal. En effet, chaque plugin est porté par un processus qui porte le framework Android. (Je sais, il y a des astuces pour partager l'essentiel du framework en mémoire par les différentes applications).

Lors de l'utilisation de plugin, il est plus judicieux de partager le processus entre l'application racine et les plugins. C'est très facile à faire sous Android. Juste quelques paramètres dans les manifests.

Dans le manifest de l'application racine, il faut indiquer un **sharedUserId** dans le marqueur **<manifest/>**.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="fr.prados.pluginview.client"
    android:versionCode="1"
    android:versionName="1.0"
    android:sharedUserId="fr.prados.pluginview"
    >
```

Puis dans le marqueur **<application/>** il faut donner un nom au processus via l'attribut **android:process**.

```
<application
  android:process="fr.prados.pluginview"
```

Il faut procéder de même pour les applications portant les plugins.

Cela présente plusieurs avantages. La communication avec les plugins s'effectue dans le même processus, sans passer par le noyau Linux. Les méthodes **asInterface()** présentée plus haut utilisent le cast et non la création d'un proxy. Ainsi, les méthodes ne peuvent plus retourner des **RemoteExceptions**. La découverte des plugins et leurs connections est plus rapide. Il n'est plus nécessaire d'exposer les services aux autres applications. Un attribut **export="false"** pour les services des plugins permet de garantir la sécurité.

8 Pour finir

Nous voici arrivé au terme de ce voyage dans une fonctionnalité peu connue d'Android. Tous les sources sont disponibles. Nous avons proposé une petite classe pour découvrir et gérer les connexions avec les plugins ; nous avons créé des layouts dans les plugins pour les utiliser dans une autre application ; puis nous avons optimisé et sécurisé tout cela.

Cela semble compliqué, mais en fait, avec une classe et quelques classes génériques présentes dans les sources pour simplifier encore la vie, en dix minutes vous pouvez avoir des plugins graphiques. ■

C++ STANDARD LIBRARY / STL REPARTONS SUR DE BONNES BASES

par Laurent NAVARRO [Développeur / formateur freelance - Altidev Toulouse]

Tous les compilateurs C++ sont normalement livrés avec la librairie standard C++ telle que définie dans la norme. Cependant elle souffre d'une certaine désaffection de la part des développeurs, remise en avant avec la norme C++ 11 cette série d'articles a pour but de vous aider à repartir sur de bonnes bases avec cette librairie aussi bien en C++ 98 qu'en C++ 11.

Mon constat est que la librairie standard C++ et, plus particulièrement, la STL (*Standard Template Library*), n'est pas forcement bien connue et pas aussi utilisée qu'elle pourrait l'être, car elle a souffert à ces débuts d'implémentation plus ou moins stables, d'une documentation plutôt absconse, de notations lourdes et d'être une source de message d'erreur particulièrement incompréhensible.

Pour le dernier point, c'est toujours vrai, mais pour les autres points, les choses se sont améliorées avec C++ 11 qui a introduit quelques syntaxes qui la rende plus pratique à utiliser (*Auto*, initialiseurs, **`foreach`**, *move*, *lambda*). D'autre part la librairie standard est devenue un point focal de la programmation moderne en C++, ses fonctionnalités ont d'ailleurs significativement augmenté et les prochaines versions de la norme tendent à encore augmenter sa couverture fonctionnelle.

La vocation de ces articles n'est pas de présenter de façon exhaustives ces classes, mais seulement les fonctionnalités les plus utilisées. Voir **[CppReference]** pour une information plus exhaustive.

1 Les chaînes de caractères (string)

La classe **`string`** va vous permettre de gérer plus simplement les chaînes de caractères qu'en C classique tout en gardant la capacité de récupérer la chaîne C sous-jacente facilement.

Si vous utilisez un *framework*, vous avez déjà probablement une classe de gestion des chaînes à votre disposition (Qstring avec QT, wxString avec wxWidget, AnsiString avec C++ Builder, CString en Visual Studio) l'intérêt d'utiliser la classe **`string`** dans ce cas est assez limité, sauf si vous souhaitez écrire un morceau de code portable et indépendant dudit framework.

1.1 Avantages / Inconvénients

Quels sont les avantages de cette classe par rapport à une gestion classique en C ? Le principal intérêt est de faciliter toutes les opérations d'affectation et de concaténation de chaînes, de ne plus avoir à gérer la mémoire et ainsi de s'affranchir d'un certain nombre de problèmes et de réduire les cas de buffer overflow.

1.2 Utilisation

Plusieurs constructeurs sont disponibles, par défaut c'est une chaîne vide. Un constructeur permet de construire à partir d'une chaîne C classique ou d'une sous-chaîne. On peut aussi dupliquer *n* fois un caractère.

```
#include <iostream>
using namespace std;

int main()
{
    string s1;  // chaine vide
    string s2="Bonjour";
    string s3("les amis");
    string s4(s3,4,4); // amis
    string s5(5,'*');
    cout << "s1="<<s1 << endl;
    cout << "s2="<<s2 << endl;
    cout << "s3="<<s3 << endl;
    cout << "s4="<<s4 << endl;
    cout << "s5="<<s5 << endl;
    return 0;
}
```

nous donne la sortie suivante :

```
s1=
s2=Bonjour
s3=les amis
s4=amis
s5=*****
```

La classe **string** va nous permettre de facilement concaténer des chaînes entres-elles mais aussi avec des chaînes C (**char*** ou littéral). Le résultat d'une concaténation sera une **string**, on pourra donc enchaîner des concaténations de **char*** à partir du moment ou une des 2 premières opérandes de l'expression est une **string**.

```
string s6;
s6=s2+" "+"les "+s4;
cout << "s6="<<s6 << endl;
```

nous donne :

```
s6=Bonjour les amis
```

La classe **string** met aussi à notre disposition des opérateurs de comparaisons qui vont comparer les chaînes et non pas les pointeurs comme en C :

```
string s1("Bonjour");
if(s1=="Bonjour")
    cout<<"s1==Bonjour"<<endl;
if(s1<"Bubu")
    cout<<"s1<Bubu"<<endl;
if(s1>"Baba")
    cout<<"s1>Baba"<<endl;
if(s1!="Au revoir")
    cout<<"s1!=Au revoir"<<endl;
```

```
s1==Bonjour
s1<Bubu
s1>Baba
s1!=Au revoir
```

Nous n'abordons pas ici la gestion des ordres en fonction des alphabets, mais sachez que c'est disponible.

Concernant la manipulation des chaînes, l'opérateur **[]** vous permettra d'accéder aux caractères de façon individuelle en base 0 (Comme une chaîne C), **insert**, **replace** et **erase** permettront de modifier des morceaux de chaîne.

Une **string** a une taille effective et une capacité, elles sont généralement égales, mais si vous souhaitez construire une chaîne par morceaux, il faudra veiller à utiliser la méthode **reserve** qui permettra d'allouer une zone mémoire plus grande et ainsi permettre à l'opérateur **+=** ou **append** ou **push_back** de faire leurs opérations dans la même zone mémoire sans avoir besoin de réallouer puis copier la chaîne existante.

En terme d'interopérabilité sortante avec les chaînes C, la méthode **c_str()** permet de récupérer un pointeur en lecture sur le buffer sous-jacent. Ce dernier deviendra potentiellement invalide si vous modifiez la chaîne contenue. La méthode **data()** différait légèrement dans sa définition, mais est à présent (C++ 11) identique à **c_str()**. La méthode copy**(char* Dest,size_t Taille)** permet de copier la **string** dans un buffer dans la limite de **Taille** caractère mais sans ajouter le zéro terminal dans la chaîne C de destination.

Existe aussi **find** qui est l'équivalent de **strpos** et vous permettra de localiser une sous-chaîne.

```
string s1("Bonjour");
s1.reserve(20);
s1+=" Amis";
cout << "s1="<<s1 << endl;
s1.insert(8,"les ");
cout << "s1 post insert ="<<s1 << endl;
s1.replace(8,3,"chers");
cout << "s1 post replace ="<<s1 << endl;
s1.erase(3,9);
cout << "s1 post erase ="<<s1 << endl;
s1[5]='a';
cout << "s1 post [] ="<<s1 << endl;
printf("Chaine C = %s \n",s1.c_str());
char Buff[100]="01234567890123456789";
s1.copy(Buff,sizeof(Buff));
Buff[s1.length()]=0;
cout << "buff ="<<Buff << endl;
cout <<"Amis en position : "<<s1.find("amis")<< endl;
```

Nous donnera la sortie suivante :

```
s1=Bonjour Amis
s1 post insert =Bonjour les Amis
s1 post replace =Bonjour chers Amis
s1 post erase =Bons Amis
s1 post [] =Bons amis
Chaine C = Bons amis
buff =Bons amis
Amis en position : 5
```

C++11 ajoute quelques fonctions de conversions depuis et vers des variables numériques (entiers et flottants) qui sont bien pratiques.

Conversion vers des entiers signés : **stoi**, **stol**, **stoll** vers des non-signés **stoul**, **stoull** vers des flottants : **stof**, **stod**, **stold**.

Les versions entières permettent de spécifier une base (par défaut à 10), le préfixe **0x** n'est accepté que si on spécifie une base 16. Le second paramètre optionnel (et *NULLable*) permet de récupérer l'index du premier caractère non converti ou de la fin de la chaîne.

En cas d'erreur de conversion une exception **std::invalid_argument** est levée :

```
int x=stoi("25");
cout <<"x="<<x<<endl;
string s;
s=to_string(48);
cout <<"s="<<s<<endl;
cout<<"0x20 ,16="<<stoi("0x20",NULL,16)<<endl;
cout<<"20   ,16="<<stoi("20",NULL,16)<<endl;
cout<<"0101 ,2 ="<<stoi("0101",NULL,2)<<endl;
std::size_t n;
cout<<"44z="<<stoi("44z",&n)<<endl;
cout <<"n="<<n<<endl;
```

nous donne la sortie suivante :

```
x=25
s=48
0x20 ,16=32
20   ,16=32
0101 ,2 =5
44z=44
n=2
```

Pour ceux encore nombreux qui utilisent C++ 98 il vous faudra utiliser les bonnes vielles fonctions C ou les classes **istringstream**/**ostringstream** ou vous développez vos propres fonctions.

Ci-dessous un exemple de fonctions glanées sur le net,

```
#include <sstream>
#include <stdexcept>

template <typename T>
string NumberToString ( T Number )
{
 ostringstream ss;
 ss << Number;
 return ss.str();
}

template <typename T>
T StringToNumber ( const string &Text )
{
 istringstream ss(Text);
```

```
 T result;
 ss >> result;
 if(ss.fail())
    throw std::invalid_argument
("StringToNumber invalid number "+Text);
return result;
}

void Exemple()
{
    int x=StringToNumber<int>("25");
    cout <<"x="<<x<<endl;
    string s=NumberToString(48);
    cout <<"s="<<s<<endl;
}
```

Voila, vous avez à présent un bon aperçu de la classe **string**.

Maintenant que vous savez comment vous en servir, discutons un peu de comment ça marche dessous. A vrai dire, le standard n'impose rien sur la façon d'implémenter, cependant on retrouve certaines techniques dans plusieurs compilateurs.

La plus répandue est le COW (*Copy On Write*), dans cette implémentation, l'objet contient un pointeur vers une zone mémoire qui contient un compteur de référence et le buffer contenant la chaîne. Lorsqu'un objet modifie la chaîne, si le compteur de référence est supérieur à 1 alors une copie est effectuée et c'est cette copie qui sera modifiée.

Certaines implémentations utilisent juste un buffer sans le partager entre les copies, ce qui est plus simple mais moins performant. Cependant en C++ 11 avec l'avènement des opérations « move » l'impact est un peu réduit. De plus en environnement multi-thread le COW requiert l'usage de protections qui peuvent avoir un coûts non négligeable, rendant les solutions non-COW intéressantes sur des chaînes de tailles raisonnables.

Sur les implémentations les plus récentes s'ajoute souvent une optimisation nommée SSO (Short/Small String Optimisation), la philosophie ici consiste à dire que pour une petite chaîne, le coût d'allocation sur le tas est plus important que la gestion d'un objet de quelques octets (il est généralement considéré comme short string une taille inférieure à 10 à 20 octets suivant les implémentations). Si la chaîne est petite elle sera stockée directement dans l'objet sinon un buffer sur le tas sera utilisé.

2 Namespace et Multithreading

Avant de continuer plus en avant dans notre découverte, nous allons aborder deux éléments transverses.

Vous avez probablement remarqué la ligne suivante dans notre programme,

```
using namespace std;
```

Et vous êtes demandé à quoi elle sert et ce qu'elle implique hormis le fait que si on ne la met pas ça ne marche pas.

Une namespace (espace de nomage) est une technique qui permet de gérer les conflits de noms entre plusieurs librairies, sous réserve que ces dernières utilisent un espace de nommage différent. Toute librairie C++ de taille un peu significative qui se respecte, utilisera un espace de nommage en entourant l'ensemble de son code par :

```
namespace MonEspace { . . . }
```

qui aura pour effet de définir tous les éléments/symboles (classes, variables, fonctions, ...) dans l'espace de nommage **MonEspace** ce qui permettra de ne pas avoir de conflit, si un même symbole est définis dans un autre espace de nommage ou dans l'espace de nommage par défaut (sans nom).

Par contre, si on souhaite faire référence à un symbole situé dans un autre namespace il faut préfixer le symbole par son namespace :

```
MonEspace::MaClasse UnObjet("test ) ;
```

Un moyen d'éviter de préfixer tous les symboles consiste à importer tout ou partie des symboles de l'espace de nommage dans l'espace courant en utilisant l'instruction **using namespace MonEspace;** pour importer un espace entier, soit **using MonEspace::MaClasse;** pour importer un symbole seul. Attention le fait d'importer tout ou partie d'un espace de nommage augmente le risque de collision entre noms. Un moyen de réduire ce risque consiste à ne faire cet import que dans certains fichiers voire dans certaines fonctions.

La librairie standard C++ est placée dans l'espace de nommage **std**, donc sans utilisation de **using** il nous faudrait écrire ceci :

```
std::string s1="Bonjour";
std::cout << "s1="<<s1 << std::endl;
```

une version avec importation sélective (**cout** & **endl** globaux et **string** importé localement) sera :

```
using std::cout;
using std::endl;
void STR6()
{
    using std::string;
    string s1="Bonjour";
    cout << "s1="<<s1 << endl;
}
```

mais si vous ne craignez pas les conflits (ce qui sera le plus souvent le cas), **using namespace std;** fera l'affaire.

Abordons maintenant une autre problématique qui n'a rien à voir avec les espaces de nommage, à savoir le multi-threading et surtout les problèmes qui y sont liés.

Historiquement le C++ fonctionne en environnement multi-thread, mais cette notion n'apparaissait pas dans la librairie standard avant C++ 11. La librairie standard marchera bien dans des thread dissociés tant que les objets ne seront manipulés que par un seul thread. Si vous les manipulez depuis plusieurs thread il vous faudra protéger les accès avec des verrous.

Le cas de la classe **string** est un peu spécial, car potentiellement 2 objets pointent sur la même zone mémoire si la technique du Copy On Write est choisi pour l'implémentation (chose que vous n'êtes pas supposé savoir). En C++ 11, d'après le standard ça devrait marcher, mais pour les versions plus

anciennes, ça dépend du compilateur, de la version ou de la librairie que vous utilisez et ce n'est pas facile de savoir si ça marche bien ou pas.

Une technique vous sera peut être utile, pour être sur qu'un objet **string** est bien une copie, il suffit d'écrire :

```
MaNouvelleChaine=MaChaineSource.c_str() ;
```

Si vous diagnostiquez que votre compilateur ne traite pas les **string** d'une façon satisfaisante, vous pourrez toujours essayer de chercher des implémentations alternatives telles que « Better String » ou des implémentations alternatives de la STL (STLPort par exemple) ou tout simplement re-coder une version thread-safe simplifiée de la classe **string** pour les endroits où vous en avez besoin dans votre programme.

3 Les collections séquentielles : la classe vector

Nous allons parler à présent des classes génériques de gestion de collections (aussi nommés conteneurs) fournis dans la librairie standard, qui pour cette partie est généralement nommée STL (*Standard Template Library*).

Les conteneurs fournis sont un tableau dynamique, des listes chaînées, une Double Ended Queue, un tableau associatif, des ensembles. Auxquels il faut ajouter quelques algorithmes génériques pour les manipuler.

Nous étudierons une bonne partie des classes et algorithmes au fils de cette série d'articles, mais aujourd'hui nous allons nous contenter des classes conteneurs séquentiels à savoir **vector**,**deque** et **list**.

3.1 Présentation

La classe **vector** est une classe générique qui implémente un tableau unidimensionnel redimensionnable dynamiquement, dont les éléments sont stockés de façon contiguë, qui a la « conscience » de sa dimension et est capable de faire du contrôle d'indice.

Voila un petit exemple qui crée un tableau de 3 entiers et l'affiche :

```
#include<vector>
using namespace std;

vector<int> Tbl1(3);
Tbl1[1]=10;
for(unsigned i=0;i<Tbl1.size();i++)
    cout<<"["<<i<<"]="<<Tbl1[i]<<endl;
```

qui affiche ceci :

```
[0]=0
[1]=10
[2]=0
```

La déclaration nous permet de spécifier le type des données contenues dans le **vector** qui peut être n'importe quel type : un type basique (**int**, **char**, **float**, **double**), un pointeur ou même une classe.

Différents constructeurs sont disponibles, celui utilisé ici permet de spécifier la dimension du tableau et de l'initialiser avec le constructeur par défaut de la classe, soit pour un type de base la valeur 0. Il est possible de passer un second paramètre qui sera la valeur initiale des éléments du tableau. Ceci nous permet au passage de créer un tableau d'objet d'une classe qui ne dispose pas de constructeur par défaut (ce qui n'est pas possible sur un tableau C classique).

Le constructeur par défaut (sans paramètre) de **vector** construit un vecteur de dimension zéro, qui sera généralement redimensionné par la suite.

Une fois votre tableau déclaré, vous pouvez utiliser le tableau avec l'opérateur crochet **[ ]** comme un tableau C aussi bien en lecture qu'en écriture. Cet opérateur n'effectue pas de contrôle des bornes, contrairement à la méthode **at** qui permet aussi un accès en lecture/écriture, mais qui lève une exception **std::out_of_range** si vous accédez à un indice invalide.

```
Tbl1.at(2)=15;
```

Ça fait une notation un peu bizarre (un appel de méthode en *Left-Value*), mais ça marche très bien, c'est d'ailleurs plutôt cette méthode qu'il faudrait privilégier.

Ensuite, nous avons la méthode **size()** qui nous permet d'obtenir la taille du tableau, cela est pratique quand on passe un **vector** en paramètre d'une fonction, car il n'est pas nécessaire, comme en C, de passer la dimension du tableau dans un second paramètre. Son amie, la méthode **empty()** permet de savoir si le **vector** est vide ou pas.

3.2 Les itérateurs

La STL a défini un moyen générique de parcours des collections, quelque soit leur nature, il s'agit des itérateurs. En effet, dans l'exemple précédent, avec la classe **vector**, nous avons utilisé l'opérateur de parcours aléatoire **[]** et un indice séquentiel de 0 à **size()-1**, mais par exemple, les listes chaînées n'ont pas d'opérateur **[]** et les tableaux associatifs en ont un, mais l'indice n'est pas un entier séquentiel.

Tous les conteneurs de la STL nous proposent donc de les parcourir au moyen d'un itérateur comme dans l'exemple ci dessous :

```
vector<int>::iterator it;
for(it=Tbl1.begin();it!=Tbl1.end();++it)
    cout<<*it<<endl;
```

qui affiche (sans l'indice donc) :

```
0
10
15
```

L'itérateur propose une interface unifiée quelque soit le conteneur et le type qu'il contient, cependant il est lié au type du conteneur (vous ne pourrez pas utiliser un itérateur de **list** sur un **vector**) et au type contenu dans le conteneur, d'où la déclaration qui spécifie que c'est un itérateur sur un **vector** de **int**. L'itérateur est une sorte de pointeur (mais ce n'en est pas un), les développeurs C familiers de l'arithmétique des pointeurs (les autres sont autorisés à s'épargner un mal de tête) pourront faire le parallèle avec le code « style C » suivant :

```
int TableauC[3]={10,20,30};
int *Ptr;
for(Ptr=TableauC;Ptr!=TableauC+3;++Ptr)
    cout<<*Ptr<<endl;
```

Le conteneur va nous fournir un itérateur sur le début de la collection via la méthode **begin()** et un itérateur factice sur l'élément qui suit le dernier élément à travers la méthode **end()**, en fait c'est plutôt un marqueur d'itérateur invalide dans le style de NULL pour les pointeurs.

L'opérateur de pré-incrémentation (**++it**) nous permet de passer à l'élément suivant de la collection et retourne l'itérateur retourné par **end()** quand on a fini de parcourir la collection. Ça marche aussi avec l'opérateur de post-incrémentation (**it++**), mais ce dernier retourne la valeur précédente de l'itérateur, ce qui a un léger surcoût (et en C++ on aime pas gaspiller ;-)).

Ensuite nous allons dé-référencer l'itérateur via l'opérateur ***** ou **->** si le type contenu est une classe ou structure pour accéder à la valeur (en lecture et écriture).

Les opérateurs de décrémentation sont aussi disponibles dans de nombreux cas, mais ils ne sont pas le bon moyen de parcourir une collection à l'envers, il faut utiliser les **reverse_iterator** pour cela couplé aux méthodes **rbegin()** et **rend()**.

```
vector<int>::reverse_iterator rit;
for(rit=Tbl1.rbegin();rit!=Tbl1.rend();++rit)
    cout<<*rit<<endl;
```

Sachez que si le conteneur est **const**, c'est un **const_iterator** qui sera retourné par **begin**/**end**. Il existe aussi **cbegin()** et **cend()**, pour obtenir un itérateur **const** en C++ 11.

Les opérateurs arithmétiques sont disponibles sur certains itérateurs,ainsi **it+2** permet de « pointer » sur 2 éléments après la position courante.

Les itérateurs, tout comme les pointeurs, sont un moyen de désigner un élément d'une collection ou avec deux itérateurs un intervalle dans une collection.

En voilà une illustration avec le constructeur de **vector** prenant 2 itérateurs en paramètres, en considérant que **Tbl2** contient les valeurs 1,2,3,4,5,6 :

```
vector<int> Tbl3(Tbl2.begin()+1,Tbl2.begin()+3);
```

Nous obtenons **Tbl3** contenant les valeurs 2 et 3 mais pas la valeur 4 qui était pourtant désignée par **Tbl2.begin()+3**, car la logique des itérateurs veut que l'on donne en borne de fin l'itérateur qui suit le dernier élément pour être compatible avec la logique de **end()**.

Ce constructeur fonctionnant avec un itérateur, il accepte n'importe quel itérateurs, on aurait par exemple pu dire que **Tbl2** était une **deque**, voire une **list** si on n'avait pas utilisé les opérateurs **+**, car les itérateurs de listes ne l'implémentent pas.

Maintenant que nous avons vu ces itérateurs, nous pouvons aborder le problème de l'initialisation d'un vecteur pour faire des choses similaires à l'instruction C suivante :

```
int TableauC[3]={10,20,30};
```

Il y a principalement 2 réponses, suivant que l'on utilise C++ 11 ou pas.

La réponse historique utilise un tableau (constant serait logique) et le constructeur par itérateurs :

```
const int Tbl2InitVal[]={1,2,3,4,5,6};
vector<int> Tbl2(Tbl2InitVal,Tbl2InitVal+6);
```

une façon détournée mais plus générique (voire convertible en macro) d'écrire 6 est :

```
vector<int> Tbl2(Tbl2InitVal,Tbl2InitVal+siz
eof(Tbl2InitVal)/sizeof(*Tbl2InitVal));
```

En C++ 11, vous pouvez aussi utiliser les fonctions **begin()** et **end()** qui pour un conteneur (STL ou tableau C) retourne les itérateurs de début et de fin.

```
vector<int> Tbl2b(Tbl2InitVal,end(Tbl2Init
Val));
```

ici **begin** n'était pas utile,nous n'avons utilisé que **end**.

La manipulation des itérateurs étant similaire à celle des pointeurs, vous pouvez passer des pointeurs à la place des itérateurs dans de nombreux cas.

Pour revenir à notre souhait d'initialiser un **vector**, à partir de C++ 11, vous pouvez utiliser le constructeur par liste d'initialisation qui vous permet d'écrire ceci :

```
vector<int> Tbl2={1,2,3,4,5};
```

ou avec la notation d'initialisation uniforme :

```
vector<int> Tbl2{1,2,3,4,5};
```

Une autre facilité d'écriture disponible en C++ 11 est le « range-based for » alias « foreach », qui pour parcourir une collection vous évite d'avoir à manipuler directement le concept d'itérateur (même si c'est bien l'itérateur du conteneur qui est utilisé) :

```
for(auto x : Tbl1)
    cout<<x<<endl;
```

Ici, j'utilise aussi la nouveauté relative à la déduction de type avec **auto**.

Dans cet exemple la variable **x** est une copie, mais j'aurais pu la déclarer en référence pour pouvoir modifier la valeur dans le conteneur :

```
for(auto &x : Tbl1)
```

ou en référence constante si mon souci était juste d'éviter une copie d'objet (sans intérêt sur un **int**) :

```
for(const auto &x : Tbl1)
```

En C++ 11, vous pouvez aussi vous contenter d'utiliser **auto** afin de ne juste pas avoir à mentionner explicitement le type de l'itérateur, cela sera utile par exemple pour utiliser un itérateur reverse ou si dans votre traitement vous souhaitez pouvoir accéder à l'itérateur (ce qui n'est pas possible avec le range-based for) :

```
for(auto rit=Tbl1.rbegin();rit!=Tbl1.rend();++rit)
    cout<<*rit<<endl;
```

3.3 Manipulation des éléments

A présent étudions un peu les diverses opérations possibles sur un **vector** en plus de l'initialiser et d'accéder individuellement à ses éléments.

L'opérateur **=** vous permet de simplement copier un **vector** dans un autre, le **vector** cible s'adaptera en terme de dimension, et tous les membres seront copiés un à un en utilisant le constructeur de copie pour créer les objets dans le conteneur cible.

Dans la même idée, nous pouvons réfléchir sur ce que fait vraiment le code ci-dessous :

```
Tbl1bis=MulPar2(Tbl1);
```

avec cette fonction :

```
vector<int> MulPar2(vector<int> Tbl)
{
    for(unsigned i=0;i<Tbl.size();i++)
        Tbl[i]*=2;
    return Tbl;
}
```

Tbl1 va être dupliqué pour devenir **Tbl** dans la fonction **MulPar2**, puis après avoir multiplié tous les éléments par 2, le **return** va créer un objet temporaire qui va ensuite être affecté à **Tbl1bis** par l'opérateur **=**. Donc chaque objet aura été copié 3 fois et détruit 2, ce qui s'il y a beaucoup d'élément dans le tableau et que les objets sont un peu gros, va peser un peu sur les performances.

La principale différence entre un tableau C et un **vector** au niveau des appels de fonctions est que par conception le tableau C, qui est un pointeur, passe par référence, alors que le **vector** va travailler par valeur c'est à dire par copie.

Une version plus efficace en C++ 98 sera donc de travailler avec un paramètre en entrée/sortie :

```
void MulPar2ByRef(vector<int> &Tbl)
{
    for(unsigned i=0;i<Tbl.size();i++)
        Tbl[i]*=2;
}
```

utilisable ainsi :

```
Tbl1bis=Tbl1;
MulPar2ByRef(Tbl1bis);
```

la première affectation n'étant nécessaire que si on souhaite avoir 2 variables à la fin.

En C++ 11, cela est toujours vrai, mais la nécessité peut être nuancé grâce à l'apparition des Rvalues références et à la capacité de « déplacer » des variables au lieu de les copier (move assignment et constructor). Ainsi, si le but est de faire de la modification de variable **MulPar2ByRef** reste la solution la plus efficace y compris en C++11, mais si le but est de créer un nouveau **vector** avec les nouvelles valeurs,

en C++ 11 la fonction **MulPar2** est tout à fait acceptable, car les deux copies qui étaient causées par le **return** n'ont plus lieux et sont remplacées par un déplacement. Seule persiste la copie initiale du passage de paramètre par valeur, mais cette copie n'est pas inutile, car l'objet ainsi créé sera l'objet qui sera finalement déplacé dans le résultat **Tbl1bis**.

Continuons à parcourir les méthodes de la classe **vector** :

La méthode **assign** permet de réinitialiser un tableau, soit par une valeur répétée X fois, soit par une paire d'itérateur, soit via une liste d'initialisation.

Utiliser des types de plus haut niveau c'est sympa, mais parfois on a besoin d'avoir un bon vieux pointeur pour appeler un bonne vielle fonction C ou typiquement faire un appel système, en C++ 11 la méthode **data()** retournera un pointeur sur le tableau C sous-jacent. Avant C++11, il faudra prendre l'adresse du premier élément. Dans les 2 cas, vous avez le droit d'écrire sur les données, vous êtes juste contraint de travailler sur la taille définie.

Exemple de fonction C :

```
void MulPar2C(int* Tbl,unsigned Taille)
{
    for(unsigned i=0;i<Taille;i++)
        Tbl[i]*=2;
}
```

Appel C++11 :

```
MulPar2C(Tbl1.data(),Tbl1.size());
```

Appel C++98 :

```
MulPar2C(&Tbl1[0],Tbl1.size());
```

Puisque nous avons décidé de faire des accès bas niveaux à notre **vector**, nous allons un peu regarder comment marche ce vecteur et aborder le concept de capacité.

Un **vector** a une taille qui correspond au nombre d'éléments qu'il contient et une capacité qui correspond au nombre maximal d'éléments qu'il pourrait contenir sans avoir besoin de ré-allouer une zone mémoire. Étant donné que le **vector**, par définition, effectue un stockage contiguë des éléments, quand il n'y a plus de place, il doit allouer une nouvelle zone mémoire et y déplacer les objets existants. Afin de réduire l'occurrence de cette opération qui peut, comme vous vous en doutez, s'avérer coûteuse, vous pouvez réserver de l'espace à l'avance à l'aide de la méthode **reserve(int n)**, qui allouera une zone permettant d'accueillir **n** éléments. Si vous ajoutez un élément au tableau, à l'aide de **push_back** ou **insert**, et qu'il n'y a plus de place, une ré-allocation aura lieu de façon transparente. Sous GCC, par défaut, la nouvelle capacité fait 50% de plus que la capacité précédente. En cas de ré-allocation, tous les itérateurs et pointeurs faisant référence au **vector** deviennent invalides. Concernant la réserve de place, elle ne constitue qu'une réserve de mémoire, elle ne contient aucun objet construit.

Lors de la ré-allocation, les éléments existants sont déplacés, ce qui, de façon générale, se traduit par une copie des objets au nouvel emplacement mémoire suivi d'une destruction des objets dans l'emplacement précédent.

En C++ 11, les choses peuvent être différentes grâce à la sémantique de déplacement (move assignment et constructor) qui permettra de déplacer les objets au lieu de les copier. L'usage de cette sémantique de déplacement n'a de sens que pour les classes qui manipulent des pointeurs et ont implémentées un constructeur de copie afin de faire des copies profondes.

Exemple d'un constructeur par déplacement (remarquez le double **&** en paramètre) :

```
ObjCopiable ( ObjCopiable &&Src) noexcept
{
    Ptr=Src.Ptr ;
    Src.Ptr =NULL;
}
```

Petit piège complémentaire, ce constructeur ne sera utilisé que s'il est stipulé qu'il ne lève pas d'exception avec **noexcept** et que le destructeur de la classe est lui aussi spécifié **noexcept**.

La méthode **capacity()** vous retournera la capacité actuelle du **vector**. **shrink_to_fit** introduit en C++ 11 permet de calquer la capacité sur la taille du **vector**.

Maintenant que le concept de capacité n'a plus de secret pour vous, nous pouvons étudier les diverses opérations qui vont modifier notre **vector** :

push_back(x) précédemment évoqué, ajoute un élément à la fin du tableau (en causant son éventuelle ré-allocation si besoin).

La méthode **insert** permet, elle, d'insérer un élément ou ensemble d'éléments désigné par une paire d'itérateurs ou une liste d'initialisation, juste avant un endroit désigné par un itérateur. Je sens que vous auriez préféré un indice mais sachez que **begin()+5** marche bien. Cette opération aura pour effet de décaler les éléments suivants à l'aide de l'opérateur **=** et du constructeur de copie pour le dernier élément. En C++ 11 si les opérateurs de déplacements sont définis, ils seront utilisés.

```
Tbl1.insert(Tbl1.begin()+2,40);
```

En C++ 11, deux nouvelles méthodes sont disponibles, **emplace_back** va permettre d'ajouter un élément de façon un peu similaire à **push_back**, mais en construisant cet élément directement à cet emplacement (in-place) sans faire de copie ou de déplacement. **emplace** est lui l'équivalent de **insert** en mode construction in-place.

La méthode **erase** est en fait assez similaire hormis qu'elle permet d'effacer un élément ou un ensemble d'éléments via un ou une paire d'itérateurs, causant là aussi un décalage des éléments suivants. **pop_back** est un raccourci qui efface le dernier élément.

La méthode **clear** efface simplement le contenu complet de la liste.

J'en profite pour faire une petite remarque, si vous faites un **vector** de pointeurs, le fait de supprimer un élément ne libère pas automatiquement

la mémoire qui y est associée. En C++11, vous pourrez parer à cela en utilisant **unique_ptr** ainsi **vector<unique_ptr<int>>** si vous utilisez les méthodes de construction in-place comme ceci :

```
vector<unique_ptr<int>> Tbl1;
Tbl1.emplace_back(new int(15));
cout<<*Tbl1[0]<<endl;
```

ou éventuellement un **shared_ptr**. Cela devrait aussi marcher avec les **shared_ptr** de Boost.

resize permet de redimensionner le **vector** en ajoutant de nouveaux éléments ou en supprimant les éléments surnuméraires.

La dernière fonctionnalité que nous aborderons est la capacité à comparer des vecteurs entre eux, avec les opérateurs **==,!=,<,<=,>,>=**. Ces opérations vont faire des comparaisons de chacun des éléments dans l'ordre sous réserve que le type contenu implémente les opérateurs == ou < suivant le cas.

4 Les collections séquentielles : la classe deque

Le conteneur **deque** (Double Ended QUEue) est un conteneur générique qui se prête à l'ajout d'élément au début et à la fin de la collection tout en les stockant de façon plutôt contiguë. L'intérêt d'une **deque** est qu'elle peut grandir sans entraîner de recopie comme la classe **vector** tout en offrant des performances en accès aléatoire similaire. Sa principale restriction est qu'il n'est pas possible d'obtenir le tableau C sous-jacent (car ce n'est pas un tableau C ;-))

L'utilisation d'une **deque** sera très similaire à celle d'un **vector** hormis qu'il n'y aura pas :

- de gestion de la capacité (**reserve**, **capacity**),
- d'invalidation potentielle des itérateurs, ni de déplacement des données, en cas d'accroissement de la taille par le début ou la fin (mais par le milieux si),
- **data()** pour accéder au tableau sous-jacent.

Il y aura par contre en plus, **push_front**, **pop_front** et **emplace_front** qui sont similaires à leurs cousines **xxx_back** mais qui agissent sur le début du conteneur.

Un petit exemple similaire à celui des **vector** :

```
#include<deque>
using namespace std;

deque<int> Tbl1(3);
Tbl1[1]=10;
Tbl1.at(2)=15;
Tbl1.insert(Tbl1.begin()+2,40);
Tbl1.erase(Tbl1.begin()+3);
Tbl1.push_front(-10);
deque<int>::iterator it;
for(it=Tbl1.begin();it!=Tbl1.end();++it)
    cout<<*it<<endl;
```

qui donne la sortie suivante :

```
-10
0
10
40
```

5 Les collections séquentielles : la classe list

La classe **list** implémente une liste chaînée double générique. Elle se prête à l'insertion/suppression d'élément à n'importe quel endroit. La permutation d'éléments et la fusion de liste sont aussi des opérations adaptées à ce conteneur qui va jouer sur les chainages plutôt que sur la copie des valeurs. Chaque élément occupe plus de place (double chaînage en plus de la donnée).

La manipulation sera similaire à la **deque** hormis qu'il n'y aura pas d'accesseur aléatoire **[]** et **at()**, les itérateurs ne supportent pas l'arithmétique (**+**/**-** un entier). Le seul moyen d'accéder aux éléments est de les parcourir avec un itérateur.

```
#include<list>
using namespace std;

list<int> L1;
L1.push_front(10);
L1.push_back(20);
L1.push_back(40);
L1.insert(++ ++ L1.begin(),30);
L1.erase(L1.begin());
L1.push_front(-10);
list<int>::iterator it;
for(it=L1.begin();it!=L1.end();++it)
    cout<<*it<<endl;
```

Nous donne la sortie suivante :

```
-10
20
30
40
```

La méthode **splice**, permet de fusionner 2 listes en une seule de façon très efficace, puisqu'il suffit de changer le chaînage des premiers et deniers éléments.

```
list<int> L2{50,60};
L1.splice(L1.end(),L2);
```

Nous donne la sortie suivante :

```
-10
20
30
40
50
60
```

La classe **list** met aussi à notre disposition quelques algorithmes disponibles dans la STL mais avec une implémentation spécialisée pour les listes.

La méthode **reverse()** permute l'ensemble des éléments de la liste 1,2,3,4 devient 4,3,2,1.

La méthode **sort()** permet de trier la liste à l'aide de l'opérateur <, ou d'un foncteur passé en paramètre, mais nous verrons le concept de foncteur plus tard.

La méthode **unique()** supprime les éléments successifs identiques, ce qui, si la liste est triée, permet de supprimer les doublons.

La méthode **remove(X)** supprime tous les éléments égaux à **X**.

La méthode **remove_if()** supprime tous les éléments répondant à un prédicat transmis sous forme de foncteur que nous étudierons plus tard.

La méthode **merge()** permet de fusionner deux listes triées en une liste triée unique.

6 Les collections séquentielles : quel conteneur choisir

L'étude suivante vous aidera à choisir **[STLBeckmark]**.

En résumé si votre collection est de taille plutôt fixe et que vous n'avez pas trop à supprimer d'éléments au milieu, un **vector** sera adapté.

La classe **deque** sera adapté si la taille est plus variable ou que l'on souhaite attaquer par les 2 bouts du conteneur pour faire une file par exemple.

La classe **list** sera surtout indiquée si vous souhaitez effacer des éléments en plein milieu de façon très fréquente ou fréquemment fusionner des listes. La liste peut présenter de l'intérêt particulièrement si le déplacement d'éléments est coûteux (gros éléments sans sémantique de déplacement en C++ 11), sinon de façon générale elle est peu performante.

7 Les collections séquentielles : un exemple

Je vous propose à présent d'étudier un petit exemple qui se veut agnostique au conteneur, il s'agit d'écrire une petite fonction **implode** (similaire à celle de PHP) qui permet de fusionner un ensemble de chaînes en une seule avec un séparateur.

Le but est de pouvoir s'en servir ainsi (j'utilise la notation C++ 11 seulement par confort pour les initialisations, cet exemple fonctionne parfaitement en C++ 98) :

```
    list<string> Ls{"L:un","deux","trois"};
    deque<string> Ds{"D:un","deux","trois"};
    vector<string> Vs{"V:un","deux","trois"};
    cout<<"Implode Ls = "<<implode(",",Ls)<<endl;
    cout<<"Implode Ds = "<<implode(",",Ds)<<endl;
    cout<<"Implode Vs = "<<implode(",",Vs)<<endl;
```

pour obtenir cet affichage :

```
Implode Ls = L:un,deux,trois
Implode Ds = D:un,deux,trois
Implode Vs = V:un,deux,trois
```

Afin d'être indépendant du conteneur, nous allons devoir jouer avec de la généricité, mais c'est ma foi assez transparent :

```
template <class T>
string implode(string sep,const T &Ctnr)
{
    typename T::const_iterator it;
    int StrLen=0;
    string res;
    if(Ctnr.empty())
        return res;
    for(it=Ctnr.begin();it!=Ctnr.end();++it)
        StrLen+=it->size()+sep.size();
    res.reserve(StrLen);
    res+=*Ctnr.begin();
    for(it=++Ctnr.begin();it!=Ctnr.end();++it)
        {
        res+=sep;
        res+=*it;
        }
    return res;
}
```

T est donc mon type générique, qui dans mes exemples sera en fait **vector<string>**, **deque<string>** ou **list<string>**.

Je déclare mon itérateur **it**, vous remarquez le **typename** qui est requis car **T** est un type générique, en C++ 11 nous aurions pu nous passer de cette déclaration, et utiliser **auto** dans les **for**.

Le premier **if** permet de considérer le cas d'un conteneur vide qui retourne donc une chaîne vide.

Le premier **for** permet juste d'évaluer la longueur de la chaîne finale afin de pouvoir réserver l'espace dans la chaîne **res**, d'où l'usage par la suite de **+=** qui va savoir utiliser cet espace réservé.

J'initialise ensuite la chaîne avec la première chaîne du conteneur, puis j'itère à partir de la seconde valeur (**++Ctnr.begin()**) en agrégeant à chaque fois le séparateur. Je le fais en deux instructions afin d'éviter la construction d'un chaîne temporaire (**sep+*it**).

Ce code générique fonctionne, car la fonction utilise des membres communs au 3 conteneurs utilisés ici (**::iterator**, **begin**, **end**).

Vous aurez remarqué que j'ai passé le conteneur par référence afin d'éviter une copie mais que je l'ai précisé **const** pour indiquer clairement que je ne modifie par cet objet.

Une version plus concise ne gérant pas la réservation et utilisant la syntaxe C++ 11 :

```
template <class T>
string implodeB(string sep,const T &Ctnr)
{
    if(Ctnr.empty())
        return string();
    string res=*Ctnr.begin();
    for(auto it=++Ctnr.begin();it!=Ctnr.end();++it)
        res+=sep+*it;
    return res;
}
```

Il était bien sûr aussi possible de ne pas utiliser la généricité et de faire une version propre à un conteneur, par exemple à **vector**, soit en changeant juste la signature et la déclaration de l'itérateur, soit pour utiliser des notations spécifiques à un conteneur :

```
string implodeVector(string sep,const vector<string> &Ctnr)
{
    string res;
    for(unsigned i=0;i<Ctnr.size();++i)
    {
        if(i)
            res+=sep;
        res+=Ctnr[i];
    }
    return res;
}
```

Dans ces trois exemples, j'ai utilisé un conteneur en paramètre, cependant dans de nombreux algorithmes de la STL, ce sont plutôt des itérateurs qui sont utilisés. Voila la version utilisant des itérateurs :

```
template <class InputIt>
string implode_it(string sep,InputIt first, InputIt last )
{
    InputIt it=first;
    int StrLen=0;
    string res;
    for(it=first;it!=last;++it)
        StrLen+=it->size()+sep.size();
    res.reserve(StrLen);
    res+=*first;
    for(it=++first;it!=last;++it)
        {
        res+=sep;
        res+=*it;
        }
    return res;
}
```

qui s'utilise ainsi :

```
cout<<"Implode_it Vs = "<<implode_it(",",Vs.begin(),Vs.end())<<endl;
```

cela complique l'appel mais permet de travailler simplement sur un sous-ensemble du conteneur ou avec un tableau C :

```
string Tbl[3] {"C:un","deux","trois"};
cout<<"Implode_it C = "<<implode_it(",",Tbl,Tbl+3)<<endl;
```

ou en C++ 11 :

```
cout<<"Implode_it C = "<<implode_it(",",begin(Tbl),end(Tbl))<<endl;
```

Ici une version, non optimisée, de la fonction écrite dans le style de la STL :

```
template <class InputIt>
string implode_itB(string sep,InputIt first, InputIt last )
{
    string res;
    for(;first!=last;++first)
        {
        if(!res.empty())
            res+=sep;
        res+=*first;
        }
    return res;
}
```

Dans les sources de la STL, il est en effet commun, d'utiliser l'itérateur d'entrée pour parcourir le conteneur. D'où le **for** sans initialisation et le dé-référencement de **first** qui est, en fait, la valeur de l'itération courante.

Conclusion et article(s) à venir

J'espère que cette présentation vous a donné envie d'utiliser autre chose que ces bons vieux tableaux C.

Dans le prochain article (qui risque de tarder un peu à venir vu la tête de mon agenda), nous aborderons :

- les algorithmes de la STL et l'utilisation des foncteurs qui va avec,
- les autres conteneurs (**map**, **set**, ...).
- Les exceptions standard,
- et si je m'en sens le courage les stream. ■

Références

[CppReference] Site avec doc sur la librairie standard C++, avec la partie C++ 11 (et même C++14) mise en évidence : http://en.cppreference.com/

[STLBeckmark] Site effectuant une comparaisons assez poussée des différents conteneurs : http://www.baptiste-wicht.com/2012/12/cpp-benchmark-vector-list-deque/

MISE EN PLACE D'UN SYSTÈME DE TAG SUR LES OBJETS ELF

par Thierry GAYET

Lors de l'inspection d'un objet ELF (binaire, librairies dynamiques, ...), il peut être intéressant de connaître les éléments qui sont entrés dans la phase de génération (version du noyau, version du compilateur, version des binutils, tag SVN ou GIT, ...). Ces éléments peuvent servir à remonter à l'origine d'un bug et donc aider aux investigations.

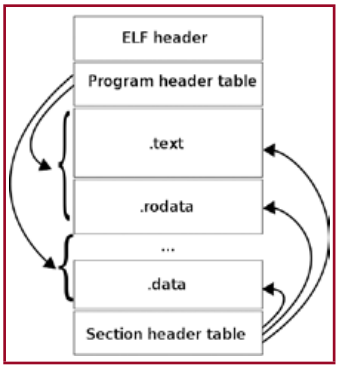
1 Les objets binaires ELF sous Linux

ELF (Executable and Linkable Format, format exécutable et liable ; anciennement Executable and Linking Format) est un format de fichier binaire utilisé pour l'enregistrement de code compilé (objets, exécutables, bibliothèques de fonctions).

Il a été développé par l'USL (Unix System Laboratories) pour remplacer les anciens formats a.out et COFF qui avaient atteint leurs limites. Aujourd'hui, ce format est utilisé dans la plupart des systèmes d'exploitation de type Unix (GNU/Linux, Solaris, IRIX, System V, BSD), à l'exception de Mac OS X.

Chaque fichier ELF est constitué d'une en-tête ELF, suivi par les données du fichier. Les données de fichier peuvent inclure :

- Un tableau d'en-tête de programme, décrivant zéro ou plusieurs segments ;



Architecture du format ELF

- Un tableau d'en-tête de section, décrivant zéro ou plusieurs sections ;
- Les données référencées par des entrées dans la table d'en-tête du programme ou le tableau d'en-tête de la section .

Les segments contiennent des informations qui sont nécessaires à l'exécution d'exécution du fichier, tandis que les sections contiennent des données importantes pour la liaison et la relocalisation. Chaque octet dans le fichier entier est pris par une section à la fois mais il peut y avoir des octets orphelins qui ne sont pas couverts par une section. Dans le cas normal d'un exécutable UNIX une ou plusieurs sections sont enfermés dans un segment.

2 Quelques commandes pour jouer avec ELF

Qu'il soit ELF32 (x86 32 bits) ou ELF64 (x86 - 64 bits), la commande file donne la carte d'identité d'un objet sous Linux :

```
$ file ./myBinElf
myBinElf: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=
0x8070f79e36c1d0b20889559eaa5a3c91efaabb8c, not stripped
$ file /lib/libcryptsetup.so.4.0.0
libcryptsetup.so.4.0.0: ELF 32-bit LSB shared object, Intel 80386, version
1 (SYSV), dynamically linked, BuildID[sha1]=0x13492859705 4b3e93f142dd94c217
9ff56e0fc55, stripped
```

ReadELF est un programme utile pour extraire les éléments de l'entête ELF :

```
$ readelf -h ./myBinElf
ELF Header:
 Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
 Class:                             ELF32
 Data:                              2's complement, little endian
 Version:                           1 (current)
 OS/ABI:                            UNIX - System V
 ABI Version:                       0
 Type:                              EXEC (Executable file)
 Machine:                           Intel 80386
 Version:                           0x1
 Entry point address:               0x8048310
```

```
Start of program headers:          52 (bytes into file)
Start of section headers:          4400 (bytes into file)
Flags:                             0x0
Size of this header:               52 (bytes)
Size of program headers:           32 (bytes)
Number of program headers:         8
Size of section headers:           40 (bytes)
Number of section headers:         29
Section header string table index: 26
```

Quelques explications sont les bienvenues. Nous pouvons remarquer quelques données au début :

```
7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
```

Ces quatre premiers octets représentent le nombre magique permettant d'identifier le type du fichier :

```
7f 45 4c 46
```

Les octets suivants représentent des meta-données du fichier comme la version, la taille, le type d'encodage des données :

```
Data:                              2's complement, little endian
Type:                              EXEC (Executable file)
```

Cela spécifie si le ELF est exécutable. Un fichier ELF peur être relogeable, un objet partage, un core file ou un fichier spécifique à un processeur. Par exemple, un driver dynamique est relogeable de part son point d'entrée.

```
Entry point address:               0x8048310
```

Le point d'entrée de l'exécutable est **`_start()`**.

Readelf permet également de lister des sections :

```
$ readelf -S ./myBinElf
There are 30 section headers, starting at offset 0x1148:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048154 000154 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048168 000168 000020 00   A  0   0  4
  [ 3] .note.gnu.build-i NOTE            08048188 000188 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        080481ac 0001ac 000020 04   A  5   0  4
  [ 5] .dynsym           DYNSYM          080481cc 0001cc 000080 10   A  6   1  4
  [ 6] .dynstr           STRTAB          0804824c 00024c 000077 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          080482c4 0002c4 000010 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         080482d4 0002d4 000030 00   A  6   1  4
  [ 9] .rel.dyn          REL             08048304 000304 000008 08   A  5   0  4
  [10] .rel.plt          REL             0804830c 00030c 000030 08   A  5  12  4
  [11] .init             PROGBITS        0804833c 00033c 00002e 00  AX  0   0  4
  [12] .plt              PROGBITS        08048370 000370 000070 04  AX  0   0 16
  [13] .text             PROGBITS        080483e0 0003e0 00020c 00  AX  0   0 16
  [14] .fini             PROGBITS        080485ec 0005ec 00001a 00  AX  0   0  4
  [15] .rodata           PROGBITS        08048608 000608 000042 00   A  0   0  4
  [16] .eh_frame_hdr     PROGBITS        0804864c 00064c 000044 00   A  0   0  4
  [17] .eh_frame         PROGBITS        08048690 000690 000104 00   A  0   0  4
  [18] .ctors            PROGBITS        08049f14 000f14 000008 00  WA  0   0  4
  [19] .dtors            PROGBITS        08049f1c 000f1c 000008 00  WA  0   0  4
  [20] .jcr              PROGBITS        08049f24 000f24 000004 00  WA  0   0  4
  [21] .dynamic          DYNAMIC         08049f28 000f28 0000c8 08  WA  6   0  4
  [22] .got              PROGBITS        08049ff0 000ff0 000004 04  WA  0   0  4
  [23] .got.plt          PROGBITS        08049ff4 000ff4 000024 04  WA  0   0  4
  [24] .data             PROGBITS        0804a018 001018 000008 00  WA  0   0  4
  [25] .bss              NOBITS          0804a020 001020 000008 00  WA  0   0  4
  [26] .comment          PROGBITS        00000000 001020 00002a 01  MS  0   0  1
  [27] .shstrtab         STRTAB          00000000 00104a 0000fc 00      0   0  1
  [28] .symtab           SYMTAB          00000000 0015f8 000460 10     29  45  4
  [29] .strtab           STRTAB          00000000 001a58 000259 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

Ainsi que les segments :

```
$ readelf --segments ./myBinElf

Elf file type is EXEC (Executable file)
Entry point 0x80483e0
There are 9 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x00120 0x00120 R E 0x4
  INTERP         0x000154 0x08048154 0x08048154 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x00794 0x00794 R E 0x1000
  LOAD           0x000f14 0x08049f14 0x08049f14 0x0010c 0x00114 RW  0x1000
  DYNAMIC        0x000f28 0x08049f28 0x08049f28 0x000c8 0x000c8 RW  0x4
  NOTE           0x000168 0x08048168 0x08048168 0x00044 0x00044 R   0x4
  GNU_EH_FRAME   0x00064c 0x0804864c 0x0804864c 0x00044 0x00044 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4
  GNU_RELRO      0x000f14 0x08049f14 0x08049f14 0x000ec 0x000ec R   0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym
.dynstr .gnu.version .gnu.version_r .rel.dyn .rel.plt .init .plt .text
.fini .rodata .eh_frame_hdr .eh_frame
   03     .ctors .dtors .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06     .eh_frame_hdr
   07
   08     .ctors .dtors .jcr .dynamic .got
```

3 Le système de TAG

En s'appuyant sur ce format dont tous les objets binaires (programmes, bibliothèques dynamiques, ...) utilisent, il est possible de rajouter une section personnalisée à ce format dynamique.

Il est donc possible de rajouter dans cette section un ensemble de données cohérent qui résumerait la chaîne de génération d'un binaire. En effet, cela peut être utile de façon a reproduire un bug dans le but de le résoudre.

Le données à inclure sont libre mais pourrait être :

- version de la toolchain (**`gcc`**, **`ld`**, ...) ;
- version de la glibc ;
- version des sources du noyau ;
- version des binutils (**`ar`**, **`strip`**, **`ranlib`**, ...) ;

- date et heure du build ;
- tag du système de gestion de versions utilisé (SVN, Mercurial, Git, ...) ;
- etc.

Cela pourrait être résumé avec un identifiant du build si sauvegarde de la carte identité des builds en base de données ;

4 Génération et pose d'un TAG avec objcopy

Un tag est simplement un fichier texte dans lequel on écrit des données importantes. Il peut donc être personnalisé à souhait :

```
$ touch ./myTag
$ echo "TOOLCHAIN = " $(TOOLCHAIN_VERSION) GCC v$ (GCC_VERSION) >> ./myTag
$ echo "TARGET       = " $(TARGET_ID) >> ./myTag
$ echo "BUILT          =  " $(shell date +"%Y/%m/%d %H:%M: %S") on
$(BUILD_HOST) $(BUILD_KIND) >> ./myTag
```

Pour faire simple, nous allons utiliser un tag avec juste une salutation :

```
echo "bonjour Thierry" > ./mytag
```

Avant d'appliquer un TAG dans les sections du header ELF, il peut être nécessaire par sécurité de supprimer un ancien TAG similaire :

```
$ objcopy --remove-section MYTAG ./myBinElf
```

Ensuite, nous pouvons ajouter notre TAG au binaire :

```
$ objcopy --add-section MYTAG=./myTag --set-section-flags
MYTAG=contents ./myBinElf
```

On peut vérifier que le tag est bien présent :

```
$ readelf -all ./myBinElf  | grep MYTAG
  [27] MYTAG             PROGBITS        00000000 00104a 000010 00   W  0   0  1
```

5 Extraction du TAG avec objcopy

Maintenant nous avons besoin d'extraire ce TAG pour pouvoir connaître son contenu. Nous allons utiliser l'outil objcopy et dd pour ce faire. De façon à simplifier l'extraction, je propose un petit script shell **extractTAG.sh** :

```
IN_F=$1
OUT_F=$2
SECTION=$3

objdump -h $IN_F |
  grep $SECTION |
  awk '{print "dd if='$IN_F' of='$OUT_F' bs=1 count=$[0x" $3 "]
skip=$[0x" $6 "]"}' |
  bash
```

Une fois le script rendu exécutable avec un **chmod +x**, son utilisation se fait de la façon suivante :

```
./extractTAG.sh  ./myBinElf  MYTAG_dump.txt  MYTAG
16+0 enregistrements lus
16+0 enregistrements écrits
16 octets (16 B) copiés, 4,2532e-05 s, 376 kB/s
$ cat ./MYTAG_dump.txt
bonjour Thierry
```

Nous retrouvons donc bien notre salutation d'origine.

Conclusion

Nous avons vu la façon de poser un tag dans dans une nouvelle section ELF d'un binaire. De même, nous avons également vu comment procéder à son extraction.

Généralisé à un système de build, cela pourrait permettre de suivre la génération d'un binaire et aussi de remonter aux contextes de génération en se basant sur les données inclus dans cette section. ■

Liens

- http://manpagesfr.free.fr/man/man5/elf.5.html
- http://pwet.fr/man/linux/formats/elf
- http://elftoolchain.sourceforge.net/for-review/libelf-by-example-20100111.pdf
- http://developers.sun.com/solaris/articles/elf.html
- http://docs.oracle.com/cd/E19082-01/819-0690/chapter6-43405/index.html
- http://www.skyfree.org/linux/references/ELF_Format.pdf
- http://www.linuxjournal.com/article/1059
- http://unixhelp.ed.ac.uk/CGI/man-cgi?objcopy+1
- http://mylinuxbook.com/readelf-command/
- http://www.bottomupcs.com/elf.html
- http://www.ensta-paristech.fr/~diam/dev/online/elf-howto-1996_07_14/ELF-HOWTO.html#toc
- http://fr.slideshare.net/varunmahajan06/gnu-linux-programstructure
- http://linuxgazette.berlios.de/issue84/hawk.html
- http://asm.sourceforge.net/articles/startup.html
- http://stackoverflow.com/questions/13908276/loading-elf-file-in-c-in-user-space
- http://linux.die.net/man/1/file
- http://linux.die.net/man/1/readelf
- http://linux.die.net/man/1/objdump

SERVEURS DÉDIÉS

— 1ER ARRIVÉ, 1ER SERVI ! —

État des stocks*

*Dans la limite des stocks disponibles au 11 octobre 2013.

PRODUITS	CPU	PROCESSEUR	RAM	DISQUE DUR	STOCK	SETUP	PRIX HT/MOIS
CRAZY FISH	Intel Xeon 3000	2 x 1,86 Ghz min.	4 Go DDR2*	250 Go SATA**	65	29€	~~29,99€~~ 19,99€
IKX-G620	Intel Pentium G620	2 x 2,6 Ghz	8 Go DDR3	250 Go SATA	14	29€	~~39,99€~~ 29,99€
IKX-CORE i3	Intel Core i3 2100T HT	2 x 2 x 2,5 Ghz	8 Go DDR3	500 Go SATA**	14	29€	~~69,99€~~ 39,99€
IKX-CORE i3 XL	Intel Core i3 530 HT	2 x 2 x 2,93 Ghz	8 Go DDR3*	1 To SATA**	19	89€	~~79,99€~~ 39,99€
IKX-CORE i7	Intel Core i7 2600 HT	4 x 4 x 3,4 Ghz	16 Go DDR3	1 To SATA	3	39€	~~99,99€~~ 59,99€
IKX-3430	Intel Xeon 3430 4 Core	4 x 2,4 Ghz	8 Go DDR3*	2 x 500 Go SATA** Raid 1 Hard	20	89€	~~189,99€~~ 89,99€
IKX-3440	Intel Xeon 3440 4 Core HT	4 x 2 x 2,53 Ghz	8 Go DDR3*	2 x 1 To SATA** Raid 1 Hard	30	89€	~~209,99€~~ 99,99€
IKX-1220L	Intel Xeon E3-1220L HT	2 x 2 x 2,2 Ghz	32 Go DDR3	2 x 500 Go SATA** Raid 1 Hard	7	89€	~~129,99€~~ 109,99€
IKX-1220	Intel Xeon E3-1220	4 x 3,1 Ghz	32 Go DDR3	2 x 1 To SATA** Raid 1 Hard	12	89€	~~149,99€~~ 119,99€
IKX-R410	Intel Bi-Xeon Quad Core HT 5000	4 x 2 x 2 Ghz	32 Go DDR3	2 x 1 To SATA** Raid 1 Hard	5	99€	~~429,99€~~ 169,99€
IKX-R710	Intel Bi-Xeon Quad Core HT E5520	4 x 2 x 2 x 2,26 Ghz	32 Go DDR3*	2 x 1 To SATA** Raid 1 Hard	1	99€	~~499,99€~~ 209,99€
IKX-R710 v2	Intel Bi-Xeon Quad Core HT X5650	4 x 2 x 2 x 2,26 Ghz	32 Go DDR3	2 x 300 Go SAS** Raid 1 Hard 2,5"	1	99€	~~519,99€~~ 249,99€
IKX-R910	Intel Quad-Xeon Hexa Core HT E7540	6 x 2 x 2 x 4 Ghz	64 Go DDR3*	2 x 300 Go SAS** Raid 1 Hard 2,5"	2	119€	~~569,99€~~ 499,99€

*Possibilité d'augmenter le niveau de RAM **Possibilité d'augmenter la taille du/des disque(s) ou d'avoir une alternative SSD/SAS

En savoir plus sur : http://express.ikoula.com

ikoula Hosting Services

01 84 01 02 50
sales@ikoula.com

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

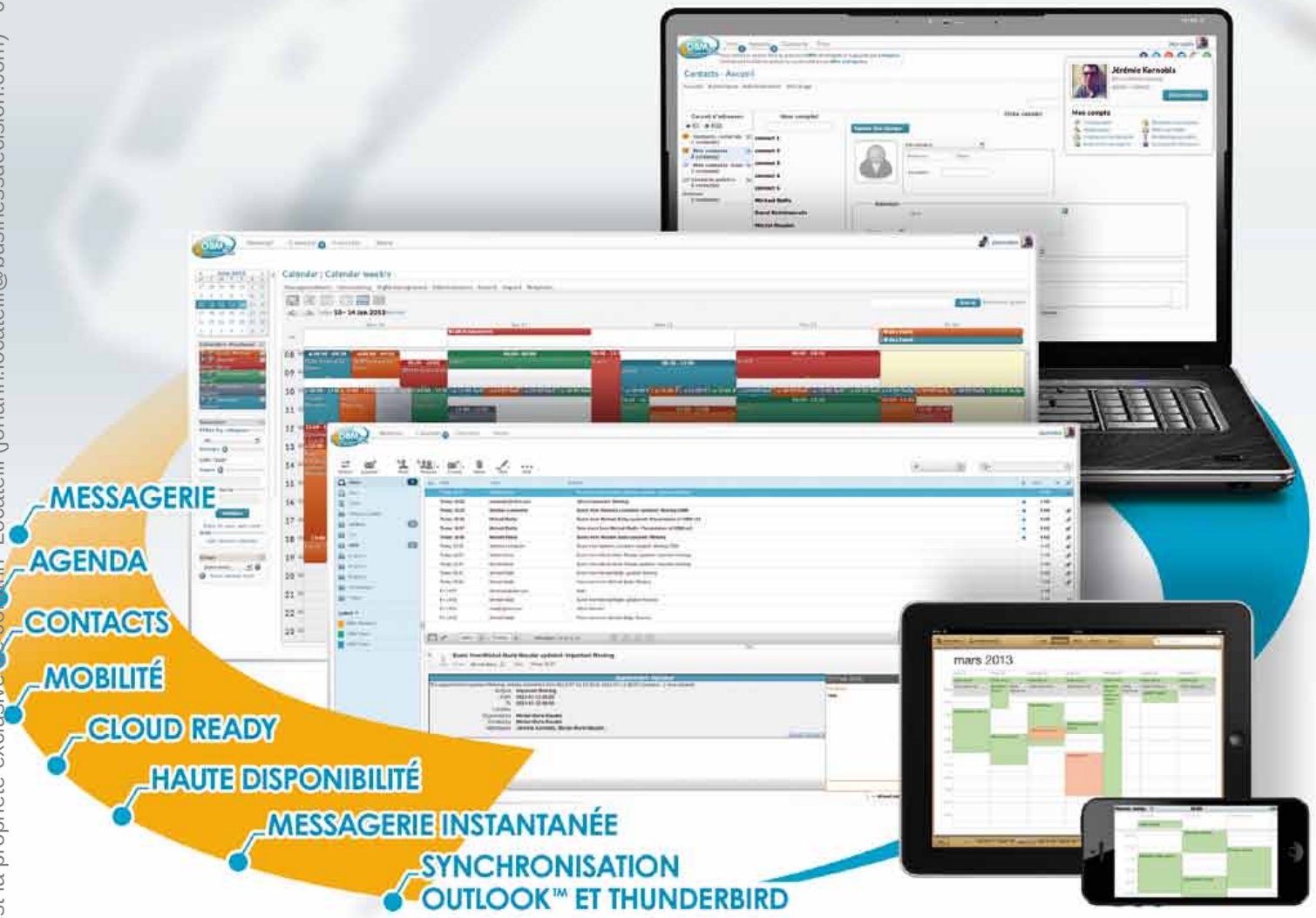
LA MESSAGERIE COLLABORATIVE

OBM.org

RÉELLEMENT LIBRE !

(SOUS LICENCE AGPL V3)

POURQUOI UTILISER LES SOLUTIONS FAUSSEMENT OPEN SOURCE ?



DÉJÀ PLUS DE 1 000 000 D'UTILISATEURS

LINAGORA

www.linagora.com

www.obm.org