N°198 novembre 2016

GNU LINUX MAGAZINE / FRANCE

DÉVELOPPEMENT ET ADMINISTRATION SUR SYSTÈMES OPEN SOURCE

SGBD / NOSQL

Optez pour le stockage optimisé de séries chronologiques avec InfluxDB p.06

TENSORFLOW / PYTHON

RÉSEAUX DE NEURONES & RECONNAISSANCE DE SYMBOLES MANUSCRITS

- Installez le framework Tensorflow
- Lancez l'apprentissage sur un jeu de données
- Reconnaissez des chiffres manuscrits

p.22

BASH / INCRON

Surveillez vos fichiers et répertoires avec inotify p.38

LIBPCAP / LANGAGE C

Codez votre programme d'analyse réseau avec la libpcap p.54

MUSIQUE / MIDI

Créez vos compositions algorithmiques musicales p.62

SYSADMIN / VLAN

Intégrez des VLANs dans une infrastructure réseau p.44

C / GUI

Programmez des menus pour vos interfaces graphiques GTK+ p.68

ET AUSSI : Nouveautés PostgreSQL - Réparez des QR codes endommagés

L 19275 - 198 - F: 7,90 € - RD

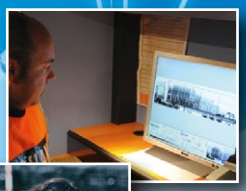
France MÉTRO. : 7,90 € – CH : 13 CHF – BEL/LUX/PORT.CONT : 8,90 € – DOM/TOM : 8,50 € – CAN : 14 $ CAD

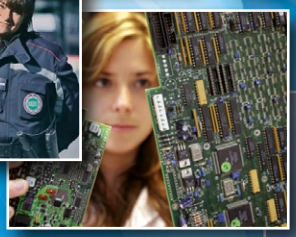
DOUANES & DROITS INDIRECTS

INFORMATICIENS

REJOIGNEZ LA DOUANE !

Mettez vos talents au service de la protection du territoire.





PROTÉGER LES CITOYENS ET L'ENVIRONNEMENT

VALIDATION

PARTICIPER AU FINANCEMENT DES SERVICES PUBLICS

RECHERCHE

ACCOMPAGNER LES ACTEURS DU COMMERCE MONDIAL

Contactez Infos Douane Service

0811 20 44 44 Service 0,06 € / min + prix appel

#LaDouaneRecrute

douane.gouv.fr / recrutement

Sur iOS et Android : douanefrance.mobi

@douane_france

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE DES FINANCES ET DES COMPTES PUBLICS

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

GNU/Linux Magazine France
est édité par Les Éditions Diamond

10, Place de la Cathédrale - 68000 Colmar - France
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 – www.ed-diamond.com

Directeur de publication : Arnaud Metzler
Chef des rédactions : Denis Bodor
Rédacteur en chef : Tristan Colombo
Responsable service Infographie : Kathrin Scali
Réalisation graphique : Thomas Pichon
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

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 €

La rédaction n'est pas responsable des textes, illustrations et photos qui lui sont communiqués par leurs auteurs. La reproduction totale ou partielle des articles publiés dans GNU/Linux Magazine France est interdite sans accord écrit de la société Les éditions Diamond. Sauf accord particulier, les manuscrits, photos et dessins adressés à GNU/Linux Magazine France, publiés ou non, ne sont ni rendus, ni renvoyés. Les indications de prix et d'adresses figurant dans les pages rédactionnelles sont données à titre d'information, sans aucun but publicitaire. Toutes les marques citées dans ce numéro sont déposées par leur propriétaire respectif. Tous les logos représentés dans le magazine sont la propriété de leur ayant droit respectif.

SUIVEZ-NOUS SUR :

https://www.facebook.com/editionsdiamond

@gnulinuxmag

LES ABONNEMENTS ET LES ANCIENS NUMÉROS SONT DISPONIBLES !

En version papier et PDF :
www.ed-diamond.com

Codes sources sur https://github.com/glmf

ÉDITORIAL

Ce mois-ci je vais revenir sur un sujet déjà abordé à de nombreuses reprises dans mes éditos : la qualité des sites internet. Nous sommes en 2016 (pratiquement en 2017) et il est encore possible d'avoir à utiliser des sites défiant toutes les règles de base de l'ergonomie ou même, pour être plus précis, qui sont mis en production alors qu'ils sont en phase de test. Je ne vais pas vous parler du blog culinaire de tel ou tel amateur, mais d'un véritable site, payé avec des fonds publics. Qu'un amateur ne soit pas au fait des aspects techniques ou juridiques peut être excusé... mais pas une entreprise professionnelle payée par une institution !

Pour ancrer mon propos dans la réalité, sachez donc qu'un beau jour, mon épouse, souhaitant acheter des billets pour un musée m'appelle à la rescousse : impossible de finaliser la commande. Je m'installe donc devant l'ordinateur et là... horreur ! Je me trouve face à la page d'authentification précédant normalement le paiement, mais dont le contenu ne correspond pas du tout puisque l'on me demande de « Choisir ma visite ». Il y a de quoi se poser des questions ! Je vérifie donc mon panier qui contient bien les tickets et je tente à nouveau de valider la commande... en vain. Je peux même admirer de multiples affichages de « Thème désactivé temporairement » ou encore, « Test expo ». Cela soulève deux problèmes :

- l'image de la ville renvoyée par le site : ouvert depuis juin 2013, ce musée possède une magnifique architecture très travaillée ; on aurait pu espérer un site internet dont justement l'architecture ait été d'une meilleure qualité et ne pas avoir à subir des pages de test !
- l'absence de mentions légales : en suivant la philosophie du logiciel libre, je me suis dit qu'il me fallait signaler les erreurs pour qu'elles soient corrigées et j'ai recherché quelle entreprise avait obtenu le marché. Ma recherche s'est avérée infructueuse pour cause d'absence de mentions légales... Pour rappel, dans le respect de la loi, celles-ci doivent être obligatoirement présentes et indiquer notamment la raison sociale, la forme juridique, le nom du responsable de la publication, les coordonnées de l'hébergeur du site, etc. Vous pourrez consulter pour une liste non exhaustive la page [1] rappelant de plus que « Le manquement à l'une de ces obligations peut être sanctionné jusqu'à un an d'emprisonnement, 75 000 € d'amende pour les personnes physiques et 375 000 € pour les personnes morales ». Nous sommes donc en présence d'un grand musée national qui a dépensé de l'argent public pour la réalisation d'un site web non fonctionnel et qui ne respecte même pas la législation en vigueur...

Ces problèmes aboutissent à un constat désolant : celui d'un musée national ayant sollicité beaucoup de ressources pour penser un contenu, mais qui bénéficie d'un site internet plus proche d'un projet en cours de développement que d'une réelle plateforme, jolie vitrine du travail des conservateurs de ce dit musée.

Comme l'aurait dit Philippe Meyer : « Nous vivons une époque moderne, le progrès fait rââââge !!! ».

[1] Site officiel de l'Administration Française, « Quelles sont les mentions légales sur un site ? » : https://www.service-public.fr/professionnels-entreprises/vosdroits/F31228

Tristan Colombo

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

ACTUELLEMENT DISPONIBLE

MISC HORS-SÉRIE N°14 !

APPRENEZ À TESTER LES VULNÉRABILITÉS DE VOS SYSTÈMES ET DE VOS SERVEURS GRÂCE À METASPLOIT

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

SOMMAIRE

GNU/LINUX MAGAZINE FRANCE N°198

abonnements

47/48 : abonnements multi-supports

53 : offres spéciales professionnelles

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

LE STOCKAGE DE SÉRIES CHRONOLOGIQUES AVEC INFLUXDB

Jérôme DELAMARCHE [Consultant indépendant en technologies Open Source – pas toutes !]

Vous devez stocker des résultats de mesures ? Échantillonnées à des moments quelconques ? Vous voulez manipuler ces données rapidement avec un langage que vous connaissez déjà ? Alors, oubliez RRDTool et stockez vos mesures dans un SGBD dont la version 1.0 va bientôt être « releasée » : InfluxDB

Mots-clés : Base de données de séries chronologiques, NoSQL

Résumé

Tout le monde est convaincu (ou devrait l'être !) que les bases de données relationnelles ne sont pas toujours idéales : en fonction de vos besoins applicatifs, vous pouvez utiliser des bases de données « orientées-documents » (comme MongoDB), « orientées-graphes » (comme Neo4J), ou « orientées-séries-chronologiques »... comme InfluxDB! Concurrent de solutions comme graphite & whisper, cette « base de données de séries chronologiques » (*Time-Series Database* ou *TSDB* en anglais) apparaît comme une solution leader [1].

Pour le stockage des métriques, l'outil le plus connu est probablement **RRDTool,** mais si vous l'utilisez, vous en connaissez les limites : nombre des données stockées limitées, valeurs moyennées en fonction de l'échelle de temps, pas de prise en compte du dimensionnement (*scalability*), pas d'API Rest native.

Les « bases de données de séries chronologiques » sont depuis apparues pour implémenter les fonctionnalités que nous attendons maintenant, à savoir : être capable de gérer des gros volumes de données (des milliers de mesures par seconde), assurer la haute disponibilité et le dimensionnement, offrir une interface REST, parler le JSON et proposer des fonctions de manipulations des données adaptées aux séries chronologiques.

InfluxDB a été développé depuis octobre 2013 en langage **Go** (apprenez-le!) par la société **influxdata [2]**. Tout d'abord basé sur le SGBD « clé-valeur » **LevelDB** de Google, il utilise dorénavant le moteur **BoltDB**, écrit lui aussi en Go.

L 'accès au service peut se faire via une API REST mais la consultation des données peut aussi être traduite dans le langage **InfluxQL** inspiré du « bon vieux » **SQL.**

1 Installation et configuration de base

Selon votre distribution Linux, un paquet **influxdb** est peut-être déjà disponible dans les dépôts. Dans le cas contraire, la page https://influxdata.com/downloads/#influxdb indique les deux commandes à exécuter. Par exemple pour du **RedHat/CentOS** :

```
$ wget https://dl.influxdata.com/influxdb/releases/influxdb-0.13.0.x86_64.rpm
$ sudo yum localinstall influxdb-0.13.0.x86_64.rpm
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Nous avons installé la dernière version stable (v0.13), la prochaine version stable sera la v1.0. Pour cet article, nous considérons que les différences de versions ne sont pas suffisamment importantes pour être mentionnées.

Que contient le paquet ? Principalement :

- le serveur **/usr/bin/influxd** ;
- le client en ligne de commande **/usr/bin/influx** ;
- un fichier de configuration nommé **/etc/influxdb/influxdb.conf** ;
- le script de service (ou le fichier pour **systemd**) pour démarrer & arrêter le service.

Avant de lancer le service, regardons brièvement sa configuration :

- le paramètre **reporting-disabled** a une valeur **false** par défaut, on peut le définir à **true** si on ne souhaite pas communiquer des infos pour aider la société InfluxData à collecter des statistiques ;
- dans la rubrique **[meta]**, le paramètre **dir** indique le répertoire où seront stockées les méta-données ;
- dans la rubrique **[data]**, le paramètre **dir** indique le répertoire où seront stockées les données ;
- la rubrique **[admin]** montre que l'interface Web d'administration est active (**enable = true**) et qu'elle écoute sur le port **8083**. Cette interface Web est minimaliste et nous utiliserons ici le mode CLI uniquement ;
- la rubrique **[http]** donne la configuration du point d'accès principal avec le serveur, donc le paramètre **enable** vaut **true**, le port d'écoute est le **8086**. Par défaut l'interface d'administration et ce service sont à l'écoute sur toutes les adresses IP du serveur. Vous pouvez modifiez ceci en fixant l'adresse IP souhaitée, par exemple **bind-address = "127.0.0.1:8086"** ;
- pour finir, vous trouverez des rubriques nommées **[[graphite]]**, **[[collectd]]**, **[[opentsdb]]** et **[[udp]]** qui correspondent à des *plugins* qui sont désactivés, car, oui, InfluxDB peut recevoir des données à ces formats (ou via UDP plutôt que TCP).

Vérifions que le serveur peut démarrer et qu'on peut s'y connecter :

```
$ sudo service influxdb start
$ influx
Visit https://enterprise.influxdata.com to register for updates,
InfluxDB server management, and monitoring.
Connected to http://localhost:8086 version 0.13.0
InfluxDB shell version: 0.13.0
> exit
```

Les choses se présentent bien, mais maintenant que faire ? Comment décrire et stocker les données ? Nous avons besoin de présenter le vocabulaire et les quelques concepts mis en œuvre par InfluxDB.

2 Représentation des données

Le serveur InfluxDB est capable de gérer plusieurs bases de données (databases). Il existe déjà une base par défaut, nommée **_internal**, contenant les méta-données internes ainsi que des données de profiling, mais nous pouvons créer notre propre base et décider d'en faire la base courante :

```
$ influx
> SHOW DATABASES
name: databases
---------------
name
_internal
> CREATE mes_mesures
> USE mes_mesures
Using database mes_mesures
>
```

Note

Les commandes sont écrites en majuscules pour des raisons de lisibilité ici. Vous pouvez les écrire en minuscules.

Note

La même opération est possible via l'API REST et serait traduite par la requête suivante :

```
$ curl -i -XPOST http://localhost:8086/query --data-urlencode
"q=CREATE DATABASE mes_mesures"
{" results ":[{}]}
```

Par analogie aux SGBD-R où les bases contiennent des tables, les bases de InfluxDB contiennent des mesures (measurements). Une mesure contient des séries de points estampillés. Cela fait beaucoup de vocabulaire, alors insérons un premier point dans notre nouvelle base. Pour cela, imaginons que nous disposons d'un capteur qui mesure la température d'une chambre froide.

L'insertion de la température relevée par ce capteur pourrait s'écrire avec une commande **INSERT** ou bien par un appel REST :

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
$ curl -i -XPOST 'http://localhost:8086/write?db=mes_mesures' --
data-binary \
    'capteurs,location=chfroide,type=temperature value=-10.3
1434055562000000000'
```

La syntaxe des données envoyées suit le Line Protocol qui est décrit par la grammaire suivante :

```
nom_mesure[,tag_key=value[...]]    field_key=field_value[,...]
[timestamp]
```

Note

Il est possible d'insérer plusieurs points dans la même requête, en séparant leurs descriptions par un caractère newline.

Dans notre cas, la valeur de **nom_mesure** est **capteurs**. Elle comporte deux tags : **location** et **type**. Elle a une seule valeur qui est **value** et l'estampille est indiquée en nombre de nano-secondes écoulées depuis le 01/01/70. Lors de l'insertion d'un point, il faut donner au moins une valeur et, si l'estampille est absente, la date courante sera utilisée.

Les tags servent de méta-données et sont indexés, ainsi que la valeur de l'estampille ; ils permettent donc d'accélérer nos futures requêtes. Les valeurs, elles, ne sont pas indexées. Les valeurs sont typées et peuvent être des chaînes, des nombres réels, des booléens (**true**, **True**, **TRUE**, **false**, ...) ou des entiers (ne pas oublier le suffixe **i** : **10i**).

Les points qui appartiennent à la même mesure et qui ont les mêmes valeurs de tags, appartiennent à la même série de points.

Vérifions le contenu de la base après cette simple insertion (on utilise le mode CLI de **influx**) :

```
> SHOW MEASUREMENTS
name: measurements
------------------
name
capteurs
> SHOW SERIES
key
capteurs,location=chfroide,type=temperature
> SELECT * FROM capteurs
name: capteurs
--------------
time                  location       type           value
1434055562000000000   chfroide       temperature    -10.3
>
```

Si vous avez suivi, vous devez vous souvenir que la première des deux requêtes suivantes va utiliser un index (car on filtre sur un tag), alors que la deuxième requête va examiner toutes les données (fort heureusement, il n'y en a qu'une !) :

```
> SELECT * FROM capteurs WHERE type = 'temperature'
name: capteurs
--------------
time                  location       type           value
1434055562000000000   chfroide       temperature    -10.3


> SELECT * FROM capteurs WHERE value < 0
name: capteurs
--------------
time                  location       type           value
1434055562000000000   chfroide       temperature    -10.3
>
```

Note

Si vous voulez obtenir des valeurs de timestamp plus « lisibles », vous pouvez utiliser la commande **precision** :

```
> PRECISION rfc3339
> SELECT * FROM capteurs WHERE type = 'temperature'
name: capteurs
--------------
time                  location       type           value
2015-06-11T20:46:02Z  chfroide       temperature    -10.3
```

3 Langage d'interrogation des données

Pour montrer les possibilités du langage InfluxQL, nous allons reprendre le jeu de données proposé par InfluxData : il s'agit des mesures de hauteurs d'eau prises à deux endroits de la côte californienne, entre le 18/08 et le 18/09/2015. Les mesures ont été réalisées toutes les 6 secondes pendant un mois.

L'insertion des données dans InfluxDB s'effectue simplement :

```
$ curl https://s3-us-west-1.amazonaws.com/noaa.water.database.0.9/
NOAA_data.txt > NOAA_data.txt
$ influx -import -path=NOAA_data.txt -precision=s
```

L'importation crée une base nommée **NOAA_water_database**. L'option **-precision=s** stipule que les estampilles sont données en secondes et non en nano-secondes.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Lançons le mode CLI et expérimentons quelques requêtes simples (ces requêtes étant triviales, leur résultat n'est pas montré ici) :

```
$ influx
> USE NOAA_water_database
> SHOW MEASUREMENTS
> SELECT * FROM h2o_feet LIMIT 3
> SELECT * FROM h2o_feet LIMIT 3 OFFSET 5
> SELECT * FROM h2o_feet ORDER BY time DESC LIMIT 3
> SELECT COUNT(water_level) AS nb FROM h2o_feet
> SELECT COUNT(water_level) FROM h2o_feet WHERE location =
'coyote_creek'
> SELECT "level description",location,water_level FROM h2o_feet
> SELECT (water_level * 2) + 4 from h2o_feet WHERE time > now() -
7d AND location =~ /coy.*/
```

Nous constatons que InfluxDB propose des fonctions de calcul numérique, des fonctions de traitement des valeurs temporelles et un support des expressions régulières (elles peuvent aussi s'appliquer aux noms des mesures, ici, c'est **h2o_feet** !).

Le traitement des données temporelles est plus intéressant : sachez que sans contrainte sur la valeur de **time**, les données retournées seront implicitement comprises entre le 01/01/70 et la valeur de **now()** !

La valeur de **time** peut être comparée avec des chaînes (voir les exemples ci-dessous), des valeurs numériques associées à une unité de temps et avec le résultat de la fonction **now()**. Il est possible de modifier la valeur de comparaison à l'aide des opérateurs plus (**+**) et moins (**-**). Les valeurs ont une unité qui peut être **w** (*week*), **d** (*day*), **h** (*hour*), **m** (*minute*), **s** (*second*), **ms** (*millisecond*) ou **u** (*microsecond*) :

```
> SELECT water_level FROM h2o_feet WHERE time > '2015-08-18
23:00:01.232000000' AND time < '2015-09-19'
> SELECT water_level FROM h2o_feet WHERE time > '2015-08-18T23:00:
01.232000000Z' AND time < '2015-09-19'
> SELECT water_level FROM h2o_feet WHERE time >
'2015-09-18T21:24:00Z' + 6m
> SELECT * FROM h2o_feet WHERE time > 1388534400s
> SELECT * FROM h2o_feet WHERE time > 24043524m + 6m
```

InfluxQL propose des fonctions d'agrégation : **COUNT()**, **DISTINCT()**, **MEAN()**, **MEDIAN()**, **SPREAD()** et **SUM()**, qui peuvent s'utiliser seules ou bien conjointement avec la clause **GROUP BY** :

```
> SELECT MEAN(water_level) FROM h2o_feet
> SELECT MEAN(water_level) FROM h2o_feet GROUP BY location
> precision rfc3339
> SELECT MEAN(water_level) FROM h2o_feet WHERE time >=
'2015-08-19T00:00:00Z' AND time <= '2015-08-27T17:00:00Z' GROUP BY
time(7d)
```

```
name: h2o_feet
--------------
time                    mean
2015-08-13T00:00:00Z    3.9866291666666687
2015-08-20T00:00:00Z    3.9117886904761847
2015-08-27T00:00:00Z    3.873656920077974

> SELECT MEAN(water_level) FROM h2o_feet WHERE time >=
'2015-07-19T00:00:00Z' AND time <= '2015-08-27T17:00:00Z' GROUP BY
time(7d) fill(0)
```

Note

L'utilisation de **GROUP BY time(...)** nécessite la présence d'une clause **WHERE** qui utilise le champ **time**. Lors du regroupement, il se peut qu'il n'y ait aucune donnée pour un intervalle particulier, dans ce cas, on peut préciser, à l'aide de la fonction **fill(...)**, la valeur par défaut à retourner !

La clause **INTO** permet de stocker les résultats d'une requête dans une autre base ou une autre mesure de la même base, par exemple, la requête suivante permet de peupler une nouvelle mesure, nommée **average**, avec la moyenne de **water_level** prise toutes les 12 minutes :

```
> SELECT mean(water_level) INTO average FROM h2o_feet WHERE
location = 'santa_monica' AND time >= '2015-08-18T00:00:00Z' AND
time <= '2015-08-18T00:30:00Z' GROUP BY time(12m)
> SELECT * FROM average
```

C'est typiquement le type de requêtes qui peut être utilisé pour réaliser des *Continuous Queries* (voir section 5.2).

InfluxDB propose les fonctions de sélection **BOTTOM()**, **FIRST()**, **LAST()**, **MAX()**, **MIN()**, **PERCENTILE()** et **TOP()** :

```
> SELECT BOTTOM(water_level,3) FROM h2o_feet
> SELECT FIRST(water_level) FROM h2o_feet WHERE location = 'santa_
monica'
```

Et pour finir, InfluxDB propose des fonctions de transformation qui appliquent un algorithme sur les données extraites afin de retourner de nouvelles valeurs. Par exemple, la fonction **DIFFERENCE()** va permettre d'obtenir la différence entre deux valeurs successives d'une même mesure :

```
> SELECT water_level FROM h2o_feet WHERE location='santa_
monica' AND time >= '2015-08-18T00:00:00Z' and time <=
'2015-08-18T00:36:00Z'
name: h2o_feet
--------------
time                    water_level
2015-08-18T00:00:00Z    2.064
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
2015-08-18T00:06:00Z      2.116
2015-08-18T00:12:00Z      2.028
...

> SELECT DIFFERENCE(water_level) FROM h2o_feet WHERE
location='santa_monica' AND time >= '2015-08-18T00:00:00Z' and
time <= '2015-08-18T00:36:00Z'
name: h2o_feet
--------------
time                      difference
2015-08-18T00:06:00Z      0.052000000000000046
2015-08-18T00:12:00Z      -0.08800000000000008
2015-08-18T00:18:00Z      0.09799999999999986
...
```

Les autres fonctions de transformation sont **DERIVATIVE()**, **ELAPSED()**, **HISTOGRAM()**, **MOVING_AVERAGE()**, **NON_NEGATIVE_DERIVATIVE()**, **STDDEV()**.

4 Un minimum d'administration

4.1 Sauvegarde et restauration

Comme tout SGBD qui se respecte, InfluxDB propose des commandes de sauvegarde et restauration.

Une sauvegarde complète nécessite de copier les métadonnées (le *Metastore*) ainsi que toutes les bases :

```
$ influxd backup /tmp/backup/
$ influxd backup -database mes_mesures /tmp/backup
```

Des options supplémentaires sont disponibles pour se connecter à un serveur distant, pour filtrer les données à sauver selon une politique de rétention (voir section 5.1) ou selon une date de départ.

La restauration utilise la commande réciproque :

```
$ influxd restore -metadir /var/lib/influxdb/meta /tmp/backup
$ influxd restore -database mes_mesures -datadir /var/lib/
influxdb/data /tmp/backup
```

Avant de relancer le service, vérifiez bien que les fichiers appartiennent à l'utilisateur **influxdb**.

4.2 Gestion des logs

Rien de bien transcendant : **influxd** écrit ses logs sur **STDERR** qui, en général, est redirigé dans un fichier de log (**/var/log/influxdb/influxd.log**). Vérifiez que **logrotate** effectuera bien une rotation de ce fichier selon vos besoins.

Notez que le fichier de configuration contient des paramètres que vous pouvez activer ou désactiver pour produire ou non des logs :

Nom de la section	Nom du paramètre booléen
[meta]	logging-enabled
[data]	query-log-enabled
[http]	log-enabled

4.3 Authentification et autorisations

Jusqu'ici, nous avons pu accéder au service sans authentification. InfluxDB propose une gestion des utilisateurs « à la SQL ». Pour activer cette fonctionnalité, il faudra créer un utilisateur d'administration puis relancer le service après avoir modifié le paramètre **auth-enabled** dans la section **[http]** du fichier de configuration.

Pour gérer les comptes des utilisateurs, les commandes à utiliser sont **CREATE USER...**, **DROP USER...**, **SHOW USERS**. Les privilèges possibles sont **READ**, **WRITE** ou **ALL** et se gèrent avec **GRANT** et **REVOKE**.

5 Notions avancées

5.1 Politique de rétention

Il est évident qu'une base de données qui stocke des mesures va voir son volume croître à l'infini ! Il est donc fort probable que vous ayez besoin de faire du ménage de temps en temps. Pour automatiser cette tâche, InfluxDB implémente des « politiques de rétention » (*Retention Policies* ou RP).

Quand nous avons créé la base **mes_mesures** et inséré un point, nous n'avons pas précisé de RP, c'est donc une politique par défaut qui a été affectée au point, comme le montre la commande suivante :

```
> SHOW RETENTION POLICIES ON mes_mesures
name      duration     shardGroupDuration    replicaN      default
default   0            168h0m0s              1             true
```

La valeur **0** indique que la durée de rétention est infinie.

Une nouvelle politique peut être créée ainsi :

```
> CREATE RETENTION POLICY un_mois ON mes_mesures DURATION 30d
REPLICATION 1
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Pour utiliser cette politique, il faut la préciser, à l'aide du paramètre **rp**, lors de l'insertion des prochains points. Par exemple :

```
$ curl -i -XPOST 'http://localhost:8086/write?db=mes_
mesures&rp=un_mois' --data-binary \
     'capteurs,location=chfroide,type=temperature value=-9.1
1434055572000000000'
```

Mais attention, maintenant, pour obtenir le point inséré, il faudra préciser la politique. Regardez la différence de résultats avec ces deux requêtes :

```
> USE mes_mesures
> SELECT * FROM capteurs;
name: capteurs
------------------------------------------------------------
time                  location       type           value
2015-06-11T20:46:02Z  chfroide       temperature    -10.3

> SELECT * FROM un_mois.capteurs;
name: capteurs
------------------------------------------------------------
time                  location       type           value
2015-06-11T20:46:12Z  chfroide       temperature    -9.1
```

Une RP peut être modifiée avec **ALTER RETENTION POLICY** ou bien supprimée avec **DROP RETENTION POLICY**.

5.2 Requêtes « en continu »

InfluxDB possède une fonctionnalité intéressante appelée *Continuous Queries* ou CQ. Le principe consiste à déclencher automatiquement et périodiquement une requête qui réalise une opération d'agrégation sur les données d'une base pour insérer les résultats éventuellement dans une autre base.

5.3 Stockage et Shards

Si vous avez été attentif, vous avez vu apparaître un paramètre nommé **shardGroupDuration** (en section 5.1). En quelques mots, il faut savoir que chaque base de données utilise des *shards* et que chaque *shard* est une instance du moteur de stockage BoltDB**.** L'intérêt est multiple, citons-en un : à chaque politique de rétention (RP) est associé un *shardGroup*. Quand les données sont obsolètes au regard de la RP, il suffit de supprimer le *shard* associé. C'est aussi pour cela qu'il faut préciser la valeur de la RP lorsqu'on requête les données !

La valeur de **shardGroupDuration** indique l'intervalle de temps à partir duquel il faudra créer un nouveau *shard*. Par exemple, si cette valeur est **7d**, alors un nouveau *shard* est créé toutes les semaines (chaque *shard* contient donc une semaine de données). Essayer la commande **SHOW SHARDS** !

6 Cluster et réplication

Au fil des changements de version, la fonctionnalité de *clusterisation* s'est vue proposée uniquement dans la version commerciale nommée **InfluxEnterprise** ! Les raisons en sont données sur le Blog de influxdata **[3]**.

Pour l'instant, la répartition de la charge et des données implique d'utiliser plusieurs instances de InfluxDB, si le besoin s'en fait sentir. Par contre, nous ne pouvons pas négliger le besoin de haute disponibilité. Une solution Open Source est proposée sous la forme d'un *Relay* **[4]**. Il s'agit d'un processus qui va relayer les écritures sur plusieurs instances de InfluxDB**.** Afin d'éviter tout SPOF (*Single-Point-Of-Failure*), il y a bien sûr deux *Relays* à placer derrière un *Load-Balancer* (de type **HA-Proxy**). Celui-ci peut aussi être configuré pour envoyer les demandes en lecture directement sur les instances **InfluxDB** et les demandes d'écriture sur les *Relays*.

Conclusion

Nous utilisons InfluxDB sur un projet de type IoT. Les performances sont excellentes, son utilisation est très simple et l'administration minimaliste. Nous apprécions ses fonctionnalités orientées « traitement de données temporelles » qui évitent aux applicatifs clients bien du travail d'agrégation inutile (voir section 3).

Le produit est toutefois en évolution constante et la partie « cluster & réplication » qui est maintenant absente du produit Open Source nécessite de passer à la version payante.

Pour finir, indiquons que la société influxdata propose d'autres produits Open Source qui forment ensemble la pile **TICK**. Cette pile est composée des quatre applications suivantes :

Telegraf qui est un agent collecteur de données diverses (système, réseau, applicatif..) et les met en forme pour les envoyer à InfluxDB (le « I » de TICK!). Les données de InfluxDB peuvent être visualisées à l'aide de **Chronograf.** Quant à **Kapacitor**, il est dédié à la gestion des alertes et à la supervision. ■

Références

[1] Bases de données de séries chronologiques : http://db-engines.com/en/ranking/time+series+dbms

[2] Site de la société InfluxData : https://influxdata.com

[3] Blog de InfluxData - fonctionnalité de clusturisation : https://influxdata.com/blog/update-on-influxdb-clustering-high-availability-and-monetization/

[4] Relay pour InfluxDB : https://github.com/influxdata/influxdb-relay/blob/master/README.md

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

POSTGRESQL 9.6 ET LE PARALLÉLISME

Guillaume LELARGE [Contributeur majeur de PostgreSQL, Consultant Dalibo, auteur du livre « PostgreSQL – Architecture et notions avancées »]
Julien ROUHAUD [Consultant Dalibo]

Le développement de la version 9.6 a commencé en juin 2015. Après de nombreux mois de travail, une première version bêta est sortie en mai 2016. Le fruit de ce travail de développements et de tests : une nouvelle version remplie de fonctionnalités intéressantes. Il est temps pour nous de connaître et de tester les principales nouveautés de cette version.

Mots-clés : PostgreSQL, Nouveautés, Parallélisation

Résumé

PostgreSQL est un moteur de base de données évoluant constamment. Avec une nouvelle version majeure chaque année, il est souvent difficile de suivre les améliorations apportées. Cet article a pour but de détailler une fonctionnalité très attendue : le parallélisme à l'exécution d'une requête.

1 Comment le parallélisme est venu

Le moteur de base de données PostgreSQL a été conçu dès le départ comme un système multi-processus, et non pas *multi-threads*. De l'avis des développeurs, il est plus simple de concevoir un système multi-processus. « Plus simple » sous-entend moins de bugs, ce qui ne peut que ravir tout le monde. PostgreSQL utilise donc plusieurs processus suivant ses besoins :

- des processus d'écriture en tâche de fond (comme le *checkpointer* et le *writer* pour les fichiers de données, ainsi que le *wal writer* pour les journaux de transactions) ;
- des processus de maintenance (le *stats collector* pour le stockage des statistiques sur l'activité, le *logger* pour la gestion des fichiers de trace, les processus *autovacuum* pour prévenir la fragmentation et pour la mise à jour des statistiques sur les données, l'*archiver* pour l'archivage des journaux de transactions, etc.) ;
- et des processus de réplication (le *wal sender* et le *wal receiver*, respectivement pour l'envoi et la réception des données de réplication).

Il utilise aussi un processus par connexion d'un client. Ce processus exécute une requête que le client vient de lui envoyer. Comme aucun des processus serveur n'utilise de *threads*, la requête est exécutée par un seul processeur (ou CPU). Si le serveur a une centaine de clients connectés, dont une grande majorité exécute des requêtes, les différents processeurs du serveur seront fortement utilisés. Cependant, si le serveur n'a que quelques clients (disons une vingtaine), qu'ils exécutent tous des requêtes, et que le serveur dispose de 128 CPU, une bonne centaine de CPUs se tourneront les pouces alors que la vingtaine d'autres travaillera d'arrache-pied. Dans ce genre de situation, on

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

aimerait bien qu'un processus exécutant une requête puisse répartir le travail sur plusieurs CPUs pour terminer plus rapidement cette exécution. On peut dire assez simplement que la vitesse d'exécution d'un CPU peut être un goulet d'étranglement pour certaines requêtes avec PostgreSQL, typiquement dans le cas de bases dites *infocentres*.

Les développeurs avaient conscience de cette limitation depuis longtemps, et ont fortement travaillé dessus depuis quelques versions. Il y avait deux possibilités : soit intégrer du *multi-threading* dans les processus PostgreSQL, soit permettre à un processus de démarrer un ou plusieurs autres processus pour les aider à l'exécution d'une requête. La première solution a été immédiatement rejetée à cause de sa complexité et du risque d'instabilité qu'elle faisait courir au serveur complet. La seconde a été retenue, et a demandé beaucoup de travail en amont afin d'ajouter l'infrastructure nécessaire pour que cela se révèle possible. Les principaux apports ont été l'ajout des *background workers* en 9.3, la modification des *background workers* pour qu'ils puissent être lancés et arrêtés à la demande en 9.4, et l'intégration de la gestion d'une mémoire partagée dynamique en 9.4.

2 Heuristiques de planification

Même si l'arrivée du parallélisme est le fruit de plusieurs années de travail, il s'agit de la toute première version ajoutant cette fonctionnalité. Comme toute fonctionnalité majeure dans PostgreSQL, son développement est itératif : la base de cette fonctionnalité est ajoutée, même si des limitations existent, puis chaque nouvelle version majeure permet de capitaliser sur cet ajout et de rendre la fonctionnalité plus riche, performante et intéressante. La première limitation sur le parallélisme concerne le type de requêtes pouvant être parallélisées. En version 9.6, seules les requêtes en lecture pourront être parallélisées. Cela exclut donc les **INSERT**, **UPDATE**, **DELETE**, **CTE** en écriture ainsi que toutes les tâches de maintenance (création d'index, **VACUUM**, etc.). Il faut également savoir que PostgreSQL est capable de ne paralléliser qu'une partie d'une requête. Par exemple, l'accès à des tables temporaires n'est pas possible de manière parallèle, mais cela n'empêchera pas une autre partie du plan d'être parallélisée.

La parallélisation se fait en lançant d'autres processus, appelés *workers*, pour traiter une partie du plan d'exécution. Ces *workers* seront pris dans la « réserve » constituée par le paramètre **max_worker_processes** (par défaut **8**). Si vous utilisez des *background workers* externes, cela diminuera d'autant le nombre de *workers* disponibles. Trouver le nombre idéal de processus supplémentaires à exécuter pour une requête précise n'est pas une chose facile, et cette première version montrera certainement quelques limites pour cela. La première limite vient du fait que ce nombre est choisi au moment de la planification, et ne prend en compte ni la charge réelle du serveur ni le nombre de requêtes parallèles concurrentes. Il n'existe non plus aucune garantie que les *workers* prévus seront disponibles au moment de l'exécution de la requête. Il est même possible de n'obtenir aucun *worker*. Il est donc primordial de configurer convenablement tous les paramètres liés à la parallélisation.

En plus de cette limite absolue du nombre de *workers* pouvant exister à un moment donné, chaque partie parallèle d'un plan ne pourra pas utiliser plus de **max_parallel_workers_per_gather** *workers*. Ce paramètre vaut actuellement **2** par défaut, mais passera à **0** lors de la sortie de la version finale, ce qui veut dire que le parallélisme ne sera pas activé par défaut.

Le calcul du nombre de processus à lancer pour une table se fait par rapport à sa taille, à l'aide du paramètre **min_parallel_relation_size** (par défaut **8** Mio). À partir de 8 Mio, un *worker* est lancé. Puis un processus supplémentaire est ajouté si la taille de la table est supérieure à trois fois ce paramètre, de manière cumulée. Une table de 216 Mio devrait donc utiliser 4 *workers*. Il est toutefois possible de surcharger table par table le nombre de *workers* à utiliser par l'intermédiaire de l'instruction SQL **ALTER TABLE**, de cette façon :

```
ALTER TABLE t1 SET (parallel_workers = 4);
```

Il est à noter que dans le cas d'une table partitionnée, le nombre de *workers* choisi est identique pour toutes les partitions, et correspond au nombre maximum trouvé grâce à l'heuristique expliquée ci-dessus. De plus, désactiver le parallélisme sur une des partitions (en configurant **parallel_workers** à **0**) désactive le parallélisme pour toutes les partitions.

Une fois le nombre de *workers* trouvé, PostgreSQL décide si utiliser un plan parallèle est rentable ou non, et si ce n'est pas le cas, un plan non parallèle sera utilisé. Ajouter des *workers* diminue bien évidemment le coût total d'un plan parallèle, mais la mise en place du parallélisme pour une

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

requête n'est pas gratuite. Il existe deux paramètres pour influencer le coût relatif d'un plan parallèle par rapport à un plan non parallèle. Le premier concerne le coût de mise en place du parallélisme (notamment la création des processus supplémentaires) : **parallel_setup_cost** (**1000** par défaut). Le second correspond au coût par ligne pour la faire transiter en mémoire partagée : **parallel_tuple_cost** (**0.1** par défaut). De plus, si une fonction est utilisée dans un plan parallèle, plus le coût estimé de la fonction (paramètre **COST** de l'ordre **CREATE FUNCTION**) est important, plus paralléliser l'exécution est intéressant. Pour un plan parallèle, PostgreSQL calculera au final un coût estimé par processus, qu'il comparera au coût d'un plan non parallèle, et choisira le meilleur.

Bien évidemment, modifier ces paramètres pour forcer des plans parallèles ou forcer un nombre élevé de *workers* pour un grand nombre de tables risque de diminuer grandement les performances des requêtes. En effet, si de nombreuses requêtes sont effectuées de manière concurrente et que trop d'entre elles sont exécutées de manière parallèle, il y aura un fort décalage entre le nombre de *workers* nécessaires pour une exécution efficace et le nombre de *workers* qu'il sera possible d'obtenir, ou le travail de tous ces processus dépassera tout simplement les capacités du serveur.

3 Premier plan parallélisé

Voyons maintenant à quoi ressemble un plan d'exécution parallèle. Pour l'instant, il n'est possible de paralléliser que les parcours séquentiels, les jointures (sauf les jointures par tri) et les agrégats. Prenons un exemple simple : la lecture d'une table et le filtre de lignes. Voici la requête exécutée :

```
SELECT * FROM t1 WHERE c1<10000;
```

Voici son plan d'exécution en 9.5 :

```
                        QUERY PLAN
-------------------------------------------------------------

 Seq Scan on t1 (actual time=0.042..1821.447 rows=9999 loops=1)
   Filter: (c1 < 10000)
   Rows Removed by Filter: 19990001
 Planning time: 0.388 ms
 Execution time: 1823.225 ms
(5 rows)
```

Et son plan d'exécution en 9.6 :

```
                        QUERY PLAN
-------------------------------------------------------------
 Gather (actual time=0.297..890.448 rows=9999 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on t1 (actual time=0.031..887.627
rows=3333 loops=3)
         Filter: (c1 < 10000)
         Rows Removed by Filter: 6663334
 Planning time: 0.121 ms
 Execution time: 893.337 ms
(8 rows)
```

La première constatation concerne le temps d'exécution. On passe de **1823** millisecondes à **893** millisecondes. Le temps d'exécution a été divisé par deux pour exactement la même requête. La deuxième constatation est le plan lui-même. L'habituel **Seq Scan** est remplacé par un **Parallel Seq Scan**, indiquant bien le fait que le parcours est parallélisé. Notez la valeur de la variable **loops** spécifiant le nombre d'exécutions du **Parallel Seq Scan**. Le nombre de lignes indiqué par le champ **rows** est à multiplier par le nombre de boucles, c'est-à-dire le nombre de *workers* plus le **Gather**. Autrement dit, il n'y a pas eu **3333** lignes récupérées, mais **9999** (**3 x 3333**). La même opération doit se faire sur le nombre de lignes supprimées par le filtre. Il est possible d'avoir le détail du traitement *worker* par *worker*, en utilisant l'option **VERBOSE** de la commande **EXPLAIN**.

Le niveau de parallélisation est indiqué par les deux lignes **Workers...** : **Workers Planned** est le nombre de processus supplémentaires choisi par le planificateur, alors que **Workers Launched** est le nombre de processus supplémentaires réellement utilisés. On pourrait en avoir moins si la configuration et l'utilisation actuelles ne permettaient pas d'en avoir plus. Donc il y a eu deux processus supplémentaires pour exécuter le parcours séquentiel. Notons qu'il s'agit bien de processus *supplémentaires*. Il y a eu donc trois processus lisant la table en même temps (le processus original et les deux processus d'aide, nommés *workers*). Le nœud **Gather** est le nœud final, qui termine la parallélisation.

Tous ces processus se voient au niveau système. Si on regarde la liste des processus pendant l'exécution du parcours parallélisé, voici ce que l'on voit :

```
$ ps -ef | grep postgres
postgres  1722 19252  0 15:18 pts/0    00:00:00 postmaster -D /opt/
postgresql/datas/r96
postgres  1738  1722  0 15:18 ?   00:00:00 postgres: checkpointer
process
postgres  1740  1722  0 15:18 ?   00:00:00 postgres: writer process
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
postgres  1741  1722  0 15:18 ?     00:00:00 postgres: wal writer
process
postgres  1742  1722  0 15:18 ?     00:00:00 postgres: autovacuum
launcher process
postgres  1743  1722  0 15:18 ?     00:00:00 postgres: stats collector
process
postgres  1781  1722 11 15:18 ?     00:00:36 postgres: postgres
postgres [local] EXPLAIN
postgres  2067  1722 54 15:23 ?     00:00:00 postgres: bgworker:
parallel worker for PID 1781
postgres  2068  1722 57 15:23 ?     00:00:00 postgres: bgworker:
parallel worker for PID 1781
```

Les processus de PID **2067** et **2068** sont les *workers* du processus original (de PID **1781**).

4 Parcours séquentiel avec plusieurs CPU

La parallélisation d'un parcours séquentiel est assez simple à comprendre. Le processus original vérifie la taille de la table à parcourir. Si cette table est plus grande que la limite configurée par le nouveau paramètre nommé **min_parallel_relation_size** (qui vaut **8** Mio par défaut), il va initialiser le mode de parallélisation. Cela sous-entend allouer de la mémoire partagée pour communiquer avec les processus dont il va demander le lancement, mais aussi demander le lancement d'un certain nombre de ces processus. Il ne les lance pas lui-même pour des raisons de stabilité du serveur. C'est le processus **postmaster** qui va s'en charger (tout comme l'*autovacuum launcher* ne crée pas lui-même des sous-processus *autovacuum worker* mais demande au *postmaster* de les créer). La mémoire partagée sera principalement utilisée par les *workers* pour mettre dans des piles de messages les lignes qu'ils retournent, que le nœud **Gather** dépilera au fur et à mesure. Elle contient également le numéro du prochain bloc de la table à lire. Chaque fois qu'un *worker* a terminé de traiter un bloc, il verrouille ce compteur en accès exclusif. Il lit le numéro du prochain bloc à traiter et l'incrémente. Puis il déverrouille ce compteur, et continue son travail sur le bloc qu'il doit traiter. Évidemment, l'utilisation d'un tel système augmente le nombre de processus travaillant sur le serveur. Les ressources, comme les processeurs et les disques, seront plus fortement utilisées, ce qui était le but au départ mais ce qui peut aussi avoir des conséquences négatives.

La parallélisation ne sera pas utilisée tout le temps. Elle n'a un intérêt que si le goulet d'étranglement d'une requête est le CPU, et qu'il reste du temps CPU disponible au moment de l'exécution. De plus, la mise en place du parallélisme (lancements de processus supplémentaires) se fait requête par requête, ce qui augmente le temps avant que la requête puisse vraiment commencer à travailler. Pour terminer, les lignes récupérées par les processus parallèles échangent leurs informations au travers d'une pile de messages en mémoire partagée, ce qui a également un coût. De ce fait, un parcours séquentiel sans filtre a peu de chance d'être exécuté avec plusieurs CPU :

```
EXPLAIN (ANALYZE, COSTS off) SELECT * FROM t1;

                          QUERY PLAN
---------------------------------------------------------------
 Seq Scan on t1 (actual time=0.019..1406.963 rows=20000000
loops=1)
 Planning time: 0.073 ms
 Execution time: 2040.796 ms
(3 rows)
```

Et si on modifie suffisamment les paramètres de coût pour forcer une exécution parallélisée, on se rend compte que ce n'est effectivement pas intéressant :

```
SET parallel_tuple_cost TO 0.005;
EXPLAIN (ANALYZE, COSTS off) SELECT * FROM t1;

                          QUERY PLAN
---------------------------------------------------------------
 Gather (actual time=0.213..5392.065 rows=20000000 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on t1 (actual time=0.021..875.184
rows=6666667 loops=3)
 Planning time: 0.043 ms
 Execution time: 6268.845 ms
(6 rows)
```

On est passé de **2** secondes (en non parallélisé) à **6,2** secondes (en parallélisé). On voit bien que la parallélisation n'apporte pas forcément un gain. Un filtre, l'utilisation d'une fonction, un calcul d'agrégat sont des opérations plus coûteuses en CPU, et de ce fait plus intéressantes pour une exécution sur plusieurs CPU.

5 Agrégat avec plusieurs CPU

La parallélisation d'un agrégat est un peu plus complexe. Voyons déjà un plan d'exécution d'une agrégation parallélisée. Voici la requête exécutée :

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
SELECT count(*), min(C1), max(C1) FROM t1;
```

Et voici le plan utilisé :

```
                                QUERY PLAN
-------------------------------------------------------------------------

 Finalize Aggregate (actual time=1766.820..1766.820 rows=1
loops=1)
   ->  Gather (actual time=1766.767..1766.799 rows=3 loops=1)
         Workers Planned: 2
         Workers Launched: 2
         ->  Partial Aggregate (actual time=1765.236..1765.236
rows=1 loops=3)
               ->  Parallel Seq Scan on t1 (actual
time=0.021..862.430 rows=6666667 loops=3)
 Planning time: 0.072 ms
 Execution time: 1769.164 ms
(8 rows)
```

Dans le cas des agrégats, le fonctionnement est à peu près identique : un **Parallel Seq Scan**, un **Gather**. Cependant, deux nouveaux nœuds apparaissent : le **Partial Aggregate** et le **Finalize Aggregate**. Le **Partial Aggregate** fait un calcul intermédiaire pour les lignes qu'il a à sa charge. Le **Finalize Aggregate** combine les résultats intermédiaires pour trouver le résultat final.

Dans le cas de fonctions d'agrégat simples comme **count()**, **min()** et **max()**, il suffit de conserver la valeur respectivement du nombre de lignes, de la valeur minimale et de la valeur maximale pour chaque sous-ensemble de lignes. Trouver la valeur finale n'est qu'un calcul ou une comparaison avec ces valeurs intermédiaires.

Dans le cas de la fonction d'agrégat **avg()** (calcul de la moyenne), ne conserver que la moyenne pour un sous-ensemble ne permettra pas d'avoir un résultat final correct. Il faut donc conserver deux valeurs : la somme des valeurs considérées et le nombre de lignes. Ces deux informations sont données par chaque nœud **Partial Aggregate** au nœud **Gather**, qui les donne enfin au nœud **Finalize Aggregate** qui fera le calcul final.

6 Jointure avec plusieurs CPU

Quant aux jointures, voici un exemple montrant les capacités de parallélisation de PostgreSQL 9.6 :

```
                                QUERY PLAN
-------------------------------------------------------------------------

Gather  (actual time=3.979..1463.835 rows=999 loops=1)
Workers Planned: 2
Workers Launched: 2
  ->  Hash Join  (actual time=487.068..1455.628 rows=333 loops=3)
      Hash Cond: (b2.id = b1.id)
      ->  Parallel Seq Scan on big2 b2  (actual
time=0.096..635.855 rows=3333333 loops=3)
      ->  Hash  (actual time=3.330..3.330 rows=999 loops=3)
          Buckets: 2048  Batches: 1  Memory Usage: 60kB
          ->  Index Scan using big_id_idx on big b1  (actual
time=0.099..2.044 rows=999 loops=3)
                Index Cond: (id < 1000)
 Planning time: 0.811 ms
 Execution time: 1465.735 ms
   (12 rows)
```

On parle de jointure parallèle car c'est le nœud **Gather** qui s'occupe de la réaliser et, dans ce cas, l'intégralité du plan est parallélisée. Dans cet exemple, les deux *workers* ne seront lancés qu'une seule fois et s'occuperont de toutes les étapes du plan. Cette jointure se fait en deux étapes, qui seront effectuées parallèlement par chaque *worker* :

- calcul de tables de hachage à partir des informations lues par un parcours d'index sur la table interne (**b1** dans notre cas). Nous trouvons ici une autre limitation : la création de la table de hachage n'est pas parallélisée. En effet, chaque *worker* (ainsi que le nœud **Gather**) va créer sa propre table de hachage complète, le parcours d'index sera donc effectué autant de fois qu'il y a de *workers* plus une fois pour le nœud **Gather** ;
- recherche d'une correspondance du hachage pour chaque ligne lue par un parcours séquentiel parallèle dans la table externe (ici **b2**).

Une condition de filtrage étant présente sur une des tables, et ce filtre étant couvert par un index, le planificateur en a déduit qu'une jointure parallélisée était intéressante. En terme de performance, le gain est appréciable : **3,1** secondes en 9.5, et **1,4** en 9.6. Si aucun index n'était présent, ou s'il n'y avait pas de condition de filtrage, chaque *worker* aurait dû parcourir l'intégralité de la table et créer une table de hachage, ce qui aurait été bien trop coûteux. Les jointures parallèles sont également possibles pour des jointures de type « Nested Loop ».

Il est important de noter qu'une jointure parallèle est différente d'une jointure du résultat de plusieurs parcours parallèles, comme dans le plan suivant :

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
                                QUERY PLAN
-------------------------------------------------------------------------
 Hash Join  (actual time=9433.458..27196.483 rows=10000000 loops=1)
   Hash Cond: (b1.id = b2.id)
  ->  Gather  (actual time=0.433..6068.146 rows=10000000 loops=1)
        Workers Planned: 2
        Workers Launched: 2
        ->  Parallel Seq Scan on big b1  (actual time=0.052..936.618
rows=3333333 loops=3)
   ->  Hash  (actual time=9431.599..9431.599 rows=10000000 loops=1)
        Buckets: 1048576  Batches: 32  Memory Usage: 23933kB
        ->  Gather  (actual time=0.427..6244.319 rows=10000000
loops=1)
              Workers Planned: 2
              Workers Launched: 2
              ->  Parallel Seq Scan on big2 b2  (actual
time=0.085..1053.717 rows=3333333 loops=3)
 Planning time: 0.652 ms
 Execution time: 27904.106 ms
   (14 rows)
```

Il s'agit d'une jointure classique, mais bénéficiant simplement de parcours parallélisés. Ceci est bien moins avantageux car il y a deux nœuds **Gather**, et par conséquent, les *workers* sont démarrés pour chacun (pour un total de quatre *workers*). La table de hachage ne sera cependant créée qu'une seule fois dans ce cas.

Conclusion

La parallélisation était l'une des fonctionnalités les plus demandées par les utilisateurs de PostgreSQL depuis de très nombreuses années. En effet, de par sa conception, certains types d'utilisation se retrouvaient peu performants par rapport à d'autres moteurs bénéficiant du parallélisme. L'ajout de cette fonctionnalité en 9.6 montre donc un nouveau tournant pour PostgreSQL: il était déjà réputé pour ses excellentes performances dans le cadre d'applications de type OLTP, il devrait briller d'ici peu pour l'utilisation avec des bases de type *infocentre*, en offrant de bien meilleures performances pour des requêtes analysant des téraoctets de données. Et nul doute que les prochaines versions apporteront également leur lot d'optimisation sur cette fonctionnalité. Mais rappelez-vous de configurer convenablement le parallélisme sur vos instances, afin d'en tirer le meilleur parti ! ■

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

AGENCE NATIONALE DE LA SÉCURITÉ DES SYSTÈMES D'INFORMATION
ANSSI

Rejoignez-nous au cœur
de la cyberdéfense française

L'ANSSI recrute près de 100 agents par an

Toutes nos offres d'emploi sont en ligne sur **www.ssi.gouv.fr**

Retrouvez toute notre actualité RH sur LinkedIn
Contact : ***recrutement@ssi.gouv.fr***

ET BEM ! DES CSS ENFIN LISIBLES !

Dr Kiss COOL [Attention au deuxième effet...]

Vous avez déjà lu de longs fichiers de css ? Il est assez difficile de s'y retrouver... Pourtant une solution simple existe : adopter des conventions de nommage permettant d'y voir plus clair. Et plutôt que d'inventer sa propre convention, autant essayer d'utiliser ce qui existe déjà.

Mots-clés : BEM, méthode, CSS, convention nommage, Web

Résumé

Les conventions de nommage permettent de produire un code facilement lisible et maintenable par tous les membres d'une même équipe ou communauté. Il paraît donc très intéressant de les mettre en place dans le cadre de projets Web qui utilisent déjà des technologies fortement laxistes. Dans le cas des css BEM propose une solution présentée dans cet article.

BEM se définit comme une méthode de développement Web basée sur une approche par « composants » : on divise les interfaces en plusieurs blocs indépendants. C'est d'ailleurs de là que BEM tire son nom : *Block - Element - Modifier*. Je parlerai plus d'une convention de nommage que d'une méthode mais, ne jouons pas sur les mots, et voyons en quoi consiste ce BEM et comment il va permettre d'obtenir réellement un code plus lisible.

1 Exemple de page

Nous nous baserons sur un exemple de page de manière à y appliquer les principes du nommage BEM. Cette page, présentée en figure 1 contient :

- une fenêtre principale ;
- des onglets ;
- une fenêtre d'affichage du contenu ;
- une image ;
- un formulaire contenant lui-même un champ de texte et un bouton de soumission.

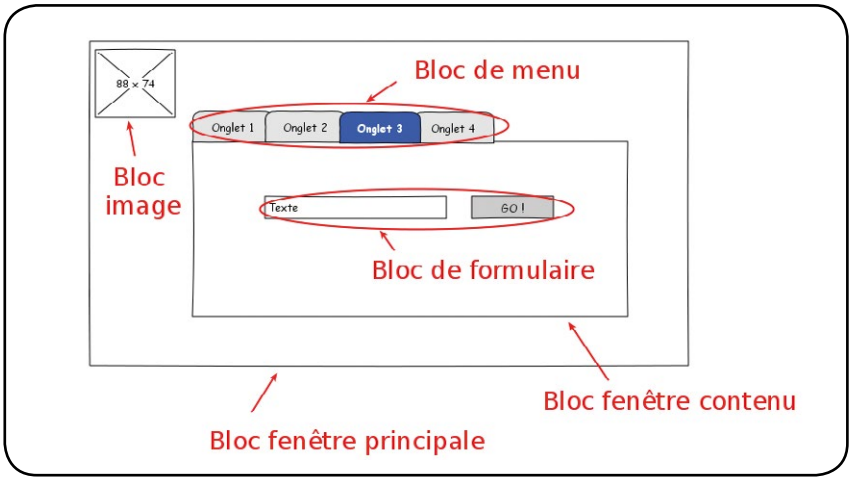


Fig. 1 : Page d'exemple

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Cette structuration permet, à l'aide des CSS, de modifier complètement l'apparence de la page. La figure 2 montre un autre agencement possible de la page d'exemple.

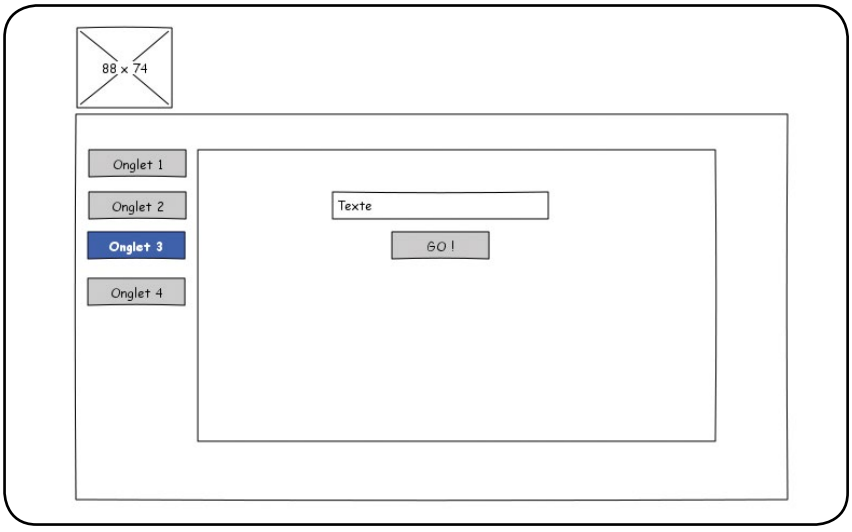


Fig. 2 : Autre agencement de la page d'exemple

2 Blocks, Elements et Modifiers

Dans notre exemple nous retrouvons des *blocks*, des *elements* et des *modifiers* qui sont appelés de manière générique des entités BEM.

2.1 Blocks

Un *block* est un élément au sens html qui peut être placé de manière arbitraire sur la page, peut être utilisé plusieurs fois et peut être inclus dans un autre *block*. Dans notre exemple, comme indiqué sur la figure 1 nous avons :

- un *block* de fenêtre principale qui contient :
  - → un *block* image ;
  - → un *block* de menu (les onglets) ;
  - → un *block* de contenu comprenant lui-même ;
  - → un *block* de formulaire.

2.2 Elements

Un *element* est un composant intégré à un *block* et qui ne peut pas être utilisé à l'extérieur de celui-ci (bien qu'il puisse être utilisé plusieurs fois). Typiquement, nos onglets sont des *elements*.

2.3 Modifiers

Un *modifier* définit l'apparence d'un *block* ou d'un *element*. De plus, un *modifier* peut changer au cours de l'exécution. C'est le cas, par exemple, sur les onglets qui peuvent devenir actifs (changement de couleur, passage au premier plan) lors du clic d'un utilisateur.

3 Convention de nommage

Nous allons maintenant enfin aborder les conventions de nommage en les appliquant directement à notre exemple. Finis les css illisibles !

Nous avons vu que notre page possède cinq blocs différents. Les classes associées à un bloc doivent être écrites en minuscules. Pour la séparation de termes, vous pouvez employer l'écriture en *camelCase* ou la séparation par des tirets. La convention de nommage définie sur https://en.bem.info/methodology/naming-convention/ ne recommande pas un style plutôt qu'un autre. Enfin une convention qui respecte le développeur ! On est loin des « PEP » Python où on nous dit précisément qu'il faut que les identifiants contiennent au minimum trois caractères, etc. Dans le cadre de cet article, puisque nous avons affaire à une convention qui nous laisse enfin libres de nos choix, j'utiliserai la notation *camelCase*. Voici une partie du squelette de la CSS définissant les *blocks* de notre exemple :

```
.window {
}

.image {
}
```

Il est intéressant de noter ici que ces simples règles, en utilisant BEM, définissent déjà des choix conceptuels : la classe **image** est utilisée pour un *block* et pourra être utilisée ailleurs. Nous aurions pu choisir de considérer qu'il s'agissait d'un *element* de **window**. À ce moment-là, nous aurions séparé le nom du *block* parent et de l'*element* par **__** :

```
.window {
}

    .window__image {
    }
```

En ce qui concerne le menu, il s'agit d'un *element* de **window**... mais les onglets sont des *elements* du **menu**. Un problème pensez-vous ? Que nenni ! BEM ne « recommande » pas d'utiliser des *elements* à l'intérieur d'autres *elements*. Il ne s'agit que d'une recommandation

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

et on voit bien que BEM a bien compris quel était l'esprit du développement Web : **JavaScript**, **html**, **php**, développeurs/graphistes, graphistes/développeurs, etc. Notez encore une fois l'aspect laxiste de la convention. Pour revenir à nos *elements* il n'est donc pas recommandé (attention, c'est différent de « déconseillé ») d'opter pour l'écriture suivante :

```
.window {
}
...
    .window__menu {
    }

        .window__menu__tab {
        }
```

Comment représenter alors cette structure ? On peut penser à nommer nos règles de la manière suivante :

```
.window {
}
...
.menu {
}

    .menu__tab {
    }
```

Mais nous perdons le lien d'inclusion de **menu** dans **window**... Les esprits chagrins diront que rien n'est clairement indiqué. Mais en fait chaque développeur pourra faire son choix, ce qui est vraiment une grande avancée pour une convention ! Regardez du côté de ces conventions dictatoriales censées homogénéiser les nommages et appréciez la souplesse de BEM ! On peut donc décider que tous les noms de *blocks* seront écrits en minuscules et que le *camelCase* indiquera un lien de parenté. On obtient :

```
.window {
}
...
    .windowMenu {
    }

        .windowMenu__tab {
        }
```

Imaginez maintenant le code pour définir le style de la zone de texte incluse dans le formulaire qui est inclus dans la zone de contenu elle-même incluse dans la fenêtre principale. Oui, c'est long et vous pensez que ce sera peu lisible avec tout cet enchevêtrement des éléments... mais c'était sans compter BEM ! Nous pourrions écrire :

```
.window {
}
...
    .windowContent {
    }

        .windowContentForm {
        }

            .windowContentForm__text {
            }
```

Comme vous le voyez chacun pourra choisir la solution qu'il juge la plus satisfaisante. Hourra pour BEM !

Passons aux onglets : un onglet peut être dans l'état « actif » (bleu, sélectionné) ou « passif » (grisé). Cette fonctionnalité fait intervenir les *modifiers* qui sont indiqués à l'aide d'un simple caractère underscore :

```
.menu {
}

    .menu__tab {
        color : gray;
    }

        .menu__tab_active {
            color : blue;
        }
```

Vous n'êtes pas convaincus ? Alors il faut que je vous parle des dernières lignes de la page décrivant la convention de nommage...

4 Pourquoi se contenter d'une unique convention ?

Bienvenue dans le monde du vrai Web ! Les dernières pages de la convention BEM nous indiquent que celle-ci peut se décliner sous différentes formes :

- nom des entités en *camelCase*, séparation *block*/*element* par un **-** et *modifiers* signalés par **--** ;
- nom des entités en minuscules (les mots sont séparés par des **-**), la séparation entre un *block* et un élément se fait par **__** et les *modifiers* sont précédés par **--** ;
- etc.

Magnifique ! BEM est pratiquement LA convention de nommage universelle. Les auteurs n'ont pas osé pousser le concept à fond mais en fait ils auraient pu écrire une convention de nommage multi-langages qui tienne en une seule ligne :

Faites ce que vous voulez !

Conclusion

Instaurer des conventions de nommage permettra toujours de rendre un code plus lisible et plus facilement maintenable, quel que soit le langage employé. Le problème est souvent de se trouver confronté à des conventions pénibles à mettre en place avec nombre de règles à respecter, des outils de flicage pour vérifier que les règles ont bien été appliquées au bon endroit, etc. Ici, cette convention a été écrite de manière à rester très proche de tous les intervenants d'un projet Web en utilisant de multiples recommandations plutôt que des règles rigides. À vous donc d'adapter les recommandations à votre guise pour créer votre propre convention de nommage, unique, qui sera appliquée par vous seul ! La classe !

Et surtout, n'oubliez pas :

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

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

DOMAINES | MAIL | **HÉBERGEMENT** | E-COMMERCE | SERVEURS

NOUVEAU ! 1&1 MANAGED CLOUD HOSTING

Le meilleur de deux mondes

Un pack d'hébergement performant associé à des ressources serveur flexibles et modulables à tout moment : **le nouveau Managed Cloud Hosting 1&1 est arrivé !** Idéal pour les projets Web les plus exigeants en termes de disponibilité, de sécurité et de flexibilité.

- ✓ **Ressources dédiées**
- ✓ **+ de 20 combinaisons de stack**
- ✓ **Géré par les experts 1&1**
- ✓ **Flexible & évolutif**
- ✓ **Prêt en moins d'1 minute**

Cloud Vendor Benchmark 2016
Rising Star Germany
EXPERTON GROUP

intel XEON inside

Trusted Performance.
Intel® Xeon® processors.

À partir de **9,99** € HT/mois (11,99 € TTC)*

1 CHOIX VOTRE DURÉE D'ENGAGEMENT
1 APPEL UN EXPERT VOUS RÉPOND
1 CERTITUDE LA PROTECTION CONTRE LES DÉFAILLANCES

0970 808 911
(appel non surtaxé)

1&1

1and1.fr

*1&1 Managed Cloud Hosting : à partir de 9,99 € HT/mois (11,99 € TTC). Pas de durée minimale d'engagement. Pas de frais de mise en service. Conditions détaillées sur 1and1.fr. 1&1 Internet SARL, RCS Sarreguemines B 431 303 775.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

APPRENTISSAGE SUPERVISÉ À L'AIDE DE RÉSEAUX DE NEURONES

Tristan COLOMBO

Les réseaux de neurones permettent de mettre en place des techniques d'apprentissage supervisés ou non. Nous allons voir comment fonctionnent ces réseaux et utiliser la bibliothèque Tensorflow pour un cas pratique de reconnaissance de chiffres manuscrits.

Mots-clés : ANN, Tensorflow, réseau de neurones artificiels, apprentissage automatique

Résumé

Le domaine de l'apprentissage automatique et des réseaux de neurones est un domaine vaste et complexe. Cet article a pour but de défricher la théorie de base et de montrer un cas concret d'application avec la reconnaissance de l'écriture manuscrite de chiffres à l'aide du *framework* Tensorflow.

L'Homme s'est inspiré de la nature pour de nombreuses découvertes : la grande bardane, plante à l'origine du velcro, la peau de requin pour les combinaisons de natation, etc. C'est ce que l'on nomme le biomimétisme. En informatiques nous avons déjà pu voir les algorithmes évolutionnistes **[1]** et je vous propose ce mois-ci de nous pencher sur les réseaux de neurones artificiels ou RNA, à ne pas confondre avec *RiboNucleic Acid* (en anglais un RNA est un ANN ou *Artificial Neural Network* et j'utiliserai cette notation pour ne pas introduire de confusion pour les lecteurs biologistes). Cet article comportera deux parties distinctes : dans un premier temps nous tenterons de comprendre sommairement le fonctionnement d'un ANN et nous en coderons une version simple puis, dans un second temps, nous nous intéresserons à un cas concret de reconnaissance de chiffres manuscrits en utilisant Tensorflow, le logiciel d'intelligence artificielle de **Google**.

1 Les réseaux de neurones artificiels

Tout part de l'observation du fonctionnement du cerveau humain qui est constitué de neurones interconnectés.

1.1 Fonctionnement d'un neurone

Un neurone est une cellule qui reçoit et envoie des signaux par impulsions électriques. Lorsque deux neurones sont connectés, naturellement la « sortie » de l'un est connectée à l' « entrée » de l'autre. Le schéma de la figure 1 illustre ce fonctionnement. L'effet de l'information véhiculée peut être soit excitateur soit inhibiteur (ce que nous traduirons assez simplement d'un point de vue mathématique...). Le signal est pondéré avant transmission ce qui permet de le moduler.

EN COUVERTURE

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

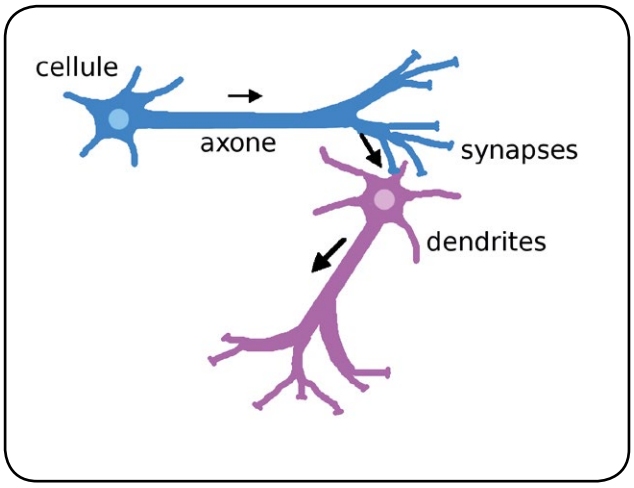


Fig. 1 : Communication entre deux neurones. L'information, représentée par les flèches, passe par la « sortie » du neurone bleu (l'axone) qui est relié à l' « entrée » du neurone violet, une dendrite, par une synapse.

Au niveau cérébral, on est en présence d'environ cent milliards de neurones qui sont de petites unités de calcul qui traitent et envoient des impulsions électriques d'environ 100mV d'une durée approximative d'1ms. Ces signaux sont envoyés vers d'autres neurones, des muscles ou des glandes.

Un neurone peut être « connecté » plusieurs fois à un même neurone et il faudra alors effectuer la somme des signaux d'entrée.

1.2 Représentation d'un neurone artificiel

En s'inspirant du fonctionnement d'un neurone « naturel », un neurone formel ou artificiel a vu le jour. Dans ce neurone on retrouve :

- des dendrites : ce sont les entrées e_i des neurones et comme la valeur de l'influx électrique est pondérée dans un véritable neurone, la valeur des entrées sera pondérée par un poids w_i. La valeur du signal d'entrée global, noté **E**, sera la somme des valeurs des entrées pondérées : $E = \sum_{i=1}^{n} w_i e_i$

On remarque ici que les outils de calcul matriciel tels que **Scilab** sont particulièrement adaptés pour ce type de calcul (voir encadré de la partie 1.3).

- des synapses : celles-ci pouvant être inhibiteur ou excitateur, une valeur de w_i négative sera inhibitrice et une valeur positive sera excitatrice.
- un seuil d'activation du neurone : celui-ci peut-être au repos ou excité. En fonction des valeurs de représentation choisies, on sera alors proche de `-1` ou `0` pour le repos et proche de `1` pour l'état excité. Le seuil d'activation est calculé par une fonction d'activation que nous noterons `f`.

Le schéma de la figure 2 montre un neurone artificiel.

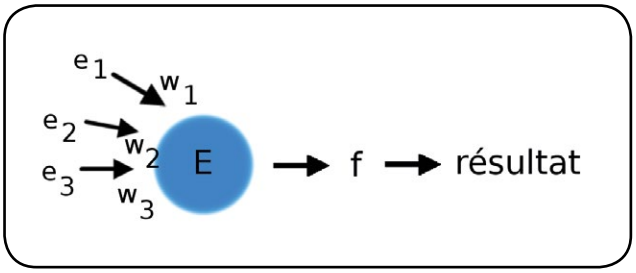


Fig. 2 : Neurone artificiel comportant trois signaux en entrées. E = w1 x e1 + w2 x e2 + w3 x e3. La valeur de sortie du neurone est f(E).

Nous commençons à distinguer comment sera structuré un neurone mais des interrogations persistent : comment déterminer la valeur des poids et comment construire `f` ?

1.3 Théorie

Pour rester dans un cadre simple, nous ne traiterons ici que le cas d'un neurone unique employé pour la classification en deux classes dans le cadre d'un apprentissage supervisé. Ce choix va répondre à l'une de nos questions, à savoir comment construire `f`. Au début des années soixante, deux sortes de réseaux monocouches ont vu le jour :

- le **Perceptron [2]** dont la fonction d'activation est :

`f(x) = 1` si `x >= 0` et `0` sinon

- et l'**Adaline** (*ADAptative LINear Element*) **[3]** dont la fonction d'activation est :

`f(x) = x`

En considérant `e` une entrée de la base d'apprentissage où chaque entrée à une dimension `d`, `R` le résultat attendu, `f` la fonction d'activation et `k` le pas d'apprentissage (`k > 0`), voici les étapes de l'apprentissage :

1. Initialisation des poids w_i par des valeurs aléatoires
2. Prendre `e`, une entrée de la base d'apprentissage
3. Calculer la valeur d'activation du neurone : $A = f\left(\sum_{i=1}^{d} w_i e_i\right)$

(Notez que l'on peut introduire l'ajout d'un biais à ce niveau comme nous le verrons dans la suite).

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

4. Calculer l'erreur sur la sortie : **delta = R - A**
5. Modifier les poids à l'étape **t** :
   $w_i(t+1) = w_i(t) + k \cdot e_i \cdot delta$
6. Répéter ces opérations un grand nombre de fois par rapport au volume des données ou répéter les opérations tant que **delta** n'est pas égal à **0** pour toutes les entrées de la base d'apprentissage.

À la fin de la phase d'apprentissage, le Perceptron aura déterminé les poids w_i qui lui permettront de calculer un résultat en accord avec la base d'apprentissage : il aura « appris » à se comporter de manière à fournir un résultat « correct ».

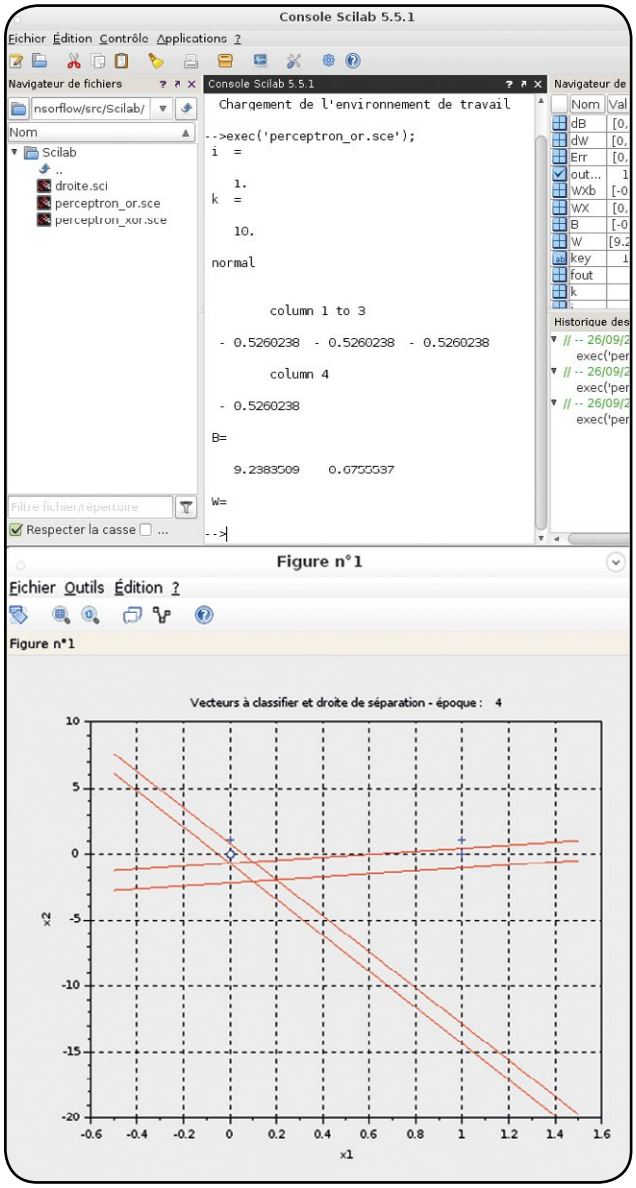


Fig. 3 : Exécution du Perceptron OR dans Scilab.

Note

Un Perceptron « OR » en Scilab

Comme nous l'avons vu en 1.2, les outils de calcul matriciel sont adaptés à la modélisation d'un Perceptron. En voici un exemple avec un OR en Scilab stocké dans un fichier **perceptron_or.sci** :

```
01: //perceptron OR
02: clear all;
03: funcprot(0);
04: exec('droite.sci'); //load the function in memory
05:
06:
07: x1min = -0.5;
08: x1max = 1.5;
09: x2min = -0.5;
10: x2max = 1.5;
11:
12: X = [0 0 1 1;0 1 0 1];
13: T = [0 1 1 1];
14:
15: i = 1
16:
17: Err = 1;
18: k=10
19: fout = 0;
20:
21: key=rand("normal");
22: key=rand("info");
23: disp(key);
24: W = rand(1,2); //initialisation aléatoire des poids
25: B = ones(1,4)*rand(1,1); //biais identique pour chaque
entrée
26:
27: while (sum(abs(Err)) ~= 0) //tant que l'erreur n'est
pas nulle
28:     //activation
29:     WX = W*X;
30:     //ajout du biais
31:     WXb = WX + B;
32:     //calcul sortie par seuillage
33:     output = WXb > fout;
34:     //calcul erreur
35:     Err = T - output;
36:     //affichage
37:     droite(X,T,W,B(1),x1min ,x1max ,x2min ,x2max ,i)
38:     i = i+1;
39:     //mise à jour pour apprentissage
40:     dW = k * Err * X';
41:     dB = sum(Err)*ones(1,4);
42:     W = W + dW;
43:     B = B + dB;
44: end
45: disp('B=',B);
46: disp('W=',W);
```

Dans Scilab il suffit ensuite de lancer : **exec('perceptron_or.sce');** pour exécuter le code (voir figure 3).

EN COUVERTURE
Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

1.4 Codage

Un Perceptron a besoin d'un ensemble d'apprentissages. Nous partirons de l'exemple suivant et nous essaierons d'obtenir une fonction qui aura le même comportement :

Label	Entrées		Sortie
Situation 1	0	0	0
Situation 2	0	1	0
Situation 3	1	0	0
Situation 4	1	1	1

Voici le code obtenu d'après les étapes présentées dans la section précédente :

```
01: from random import choice
02: from numpy import array, dot, random
03:
04: class Perceptron:
05:
06:     def __init__(self, training_data, k=0.2, bias=1):
07:         self.bias = bias
08:         self.w = random.rand(len(training_data[0][0]) + 1)
09:         self.training_data = []
10:         for e in training_data:
11:             self.training_data.append((array(e[0] + (self.
bias,)), e[1]))
12:
13:         self.k = k
14:         self.trained = False
15:
16:
17:     def f_activation(self, x):
18:         if x < 0:
19:             return 0
20:         else:
21:             return 1
22:
23:
24:     def startTraining(self, n=100, verb=False):
25:         errors = []
26:
27:         for i in range(n):
28:             e, R = choice(self.training_data)
29:             result = dot(self.w, e)
30:             delta = R - self.f_activation(result)
31:             errors.append(delta)
32:             self.w += self.k * delta * e
33:
34:         self.trained = True
35:         if verb:
36:             print(errors)
37:
38:
39:     def getFunction(self):
40:         if self.trained:
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

ACTUELLEMENT DISPONIBLE

GNU/LINUX MAGAZINE HORS-SÉRIE n°86

75 RECETTES

POUR ACCÉLÉRER VOS DÉVELOPPEMENTS !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

```
41:             return lambda *e : self.f_activation(dot(array(e +
(self.bias,)), self.w))
42:         else:
43:             return None
44:
45:
46: if __name__ == '__main__':
47:     training_data = (
48:         ((0, 0), 0),
49:         ((0, 1), 0),
50:         ((1, 0), 0),
51:         ((1, 1), 1)
52:     )
53:
54:     perceptron = Perceptron(training_data)
55:     perceptron.startTraining()
56:     fct_and = perceptron.getFunction()
57:     print('1 and 1 -> {}'.format(fct_and(1, 1)))
```

La classe **Perceptron** définit... un Perceptron. Le constructeur des lignes 6 à 14 admet trois paramètres :

- **training_data** : un ensemble d'apprentissage sous la forme d'un tuple de tuples contenant chacun un tuple des valeurs d'entrée et la valeur de sortie attendue. Les données auront donc la forme suivante :

$$
\begin{pmatrix}
((e_{11}, e_{12}, \ldots, e_{1n}), R_1),\\
\ldots\\
((e_{k1}, e_{k2}, \ldots, e_{kn}), R_k),
\end{pmatrix}
$$

- **k** : le pas d'apprentissage ;
- **bias** : on introduit un biais qui permet de décaler la fonction d'activation vers la gauche ou vers la droite. Cela permet d'effectuer un réglage pour que les prédictions correspondent au mieux aux données d'apprentissage. Ce biais est ajouté à l'ensemble d'apprentissages lors de sa conversion en liste d'**array** dans les lignes 9 à 11. Les données d'apprentissage sont alors de la forme :

$$
\begin{bmatrix}
(\text{array}[e_{11}, e_{12}, \ldots, e_{1n}, \text{bias}], R_1),\\
\ldots\\
(\text{array}[e_{k1}, e_{k2}, \ldots, e_{kn}, \text{bias}], R_k),
\end{bmatrix}
$$

L'attribut **self.w** contient des poids aléatoires de départ (ligne 8) et **self.trained** indique si le Perceptron a été entraîné ou pas (ligne 14).

Dans les lignes 17 à 21 on retrouve la fonction d'activation **f_activation()** (qui aurait très bien pu être une méthode de classe) et dans les lignes 24 à 36 la méthode d'apprentissage **startTraining()** qui suit scrupuleusement les étapes de la section 1.3. Le paramètre **n** permet de régler le nombre d'itérations et **verb** d'afficher (ou non) la liste des deltas (de manière à vérifier que l'on s'est bien arrêté avec un delta de **0**).

Enfin la fonction **getFunction()** des lignes 39 à 43 permet d'obtenir une fonction qui produira un résultat d'après les données d'entraînements. Il s'agit d'une fonction anonyme acceptant un nombre non fixé d'arguments (***e**) qui constitueront les entrées et seront multipliés par la matrice des poids obtenue. Ceci permet d'avoir une utilisation élégante du Perceptron dans le programme principal (lignes 56 et 57). On voit bien qu'il s'agit ici d'un « et » logique.

Ce qui est intéressant avec cette implémentation, c'est que la taille de la base d'apprentissage n'est pas fixée arbitrairement. On peut ainsi modéliser un **(A xor B) and C** :

```
training_data = (
    ((0, 0, 0), 0),
    ((0, 0, 1), 0),
    ((0, 1, 0), 0),
    ((0, 1, 1), 1),
    ((1, 0, 0), 0),
    ((1, 0, 1), 1),
    ((1, 1, 0), 0),
    ((1, 1, 1), 0)
)

perceptron = Perceptron(training_data)
perceptron.startTraining()
fct_xor_and = perceptron.getFunction()
print('1 xor 0 and 1 -> {}'.format(fct_xor_and(1, 0, 1)))
```

L'intérêt de l'apprentissage est ici très limité puisqu'il n'y a pas possibilité d'inférence d'une nouvelle connaissance. Les modèles de réseaux de neurones multi-couches, plus complexes, permettent cette inférence. Pour ces modèles nous n'allons pas « réinventer la roue carrée » et nous utiliserons le *framework* Tensorflow de Google.

Note

Pour le multicouche il existe d'autres algorithmes d'apprentissage : rétropropagation du gradient de l'erreur **[4]**, Sollis et Wetts **[5]**, etc.

L'exemple de cette section n'était présenté que pour comprendre les bases des réseaux de neurones. Il est bien évident qu'en l'état le programme suivant est plus efficace :

EN COUVERTURE
Ce document est la propriété exclusive de Johann Locatelli (johann.locatelli@businessdecision.com)

```
01: class Faux_Perceptron:
02:
03:     def __init__(self, training_data):
04:         self.training_data = {}
05:         for e in training_data:
06:             self.training_data[e[0]] = e[1]
07:
08:
09:     def getFunction(self):
10:         return lambda *e : self.training_data[e]
11:
12:
13: if __name__ == '__main__':
14:     training_data = (
15:         ((0, 0, 0), 0),
16:         ((0, 0, 1), 0),
17:         ((0, 1, 0), 0),
18:         ((0, 1, 1), 1),
19:         ((1, 0, 0), 0),
20:         ((1, 0, 1), 1),
21:         ((1, 1, 0), 0),
22:         ((1, 1, 1), 0)
23:     )
24:
25:     f_perceptron = Faux_Perceptron(training_data)
26:     fct_xor_and = f_perceptron.getFunction()
27:     print('1 xor 1 and 1 -> {}'.format(fct_xor_and(1, 1, 1)))
```

2 Tensorflow

Pour utiliser Tensorflow nous allons résoudre le problème suivant : étant donné un jeu d'apprentissage de chiffres en écriture manuscrite, nous souhaitons être en mesure d'observer une nouvelle image pour en prédire le chiffre. Nous nous baserons pour cela sur un tutoriel de Tensorflow **[6]** que nous commenterons et que nous modifierons.

2.1 Installation

Avant de commencer, il va nous falloir installer le *framework*. Cela passe par l'utilitaire **pip** :

```
$ export TF_BINARY_URL=https://storage.googleapis.com/tensorflow/
linux/cpu/tensorflow-0.9.0rc0-cp35-cp35m-linux_x86_64.whl
$ sudo pip3 install --upgrade $TF_BINARY_URL
```

Vous remarquez qu'il a fallu fournir une url pour installer le *framework*. Celle que j'ai utilisée correspond à Python 3.5 sur un CPU 64 bits sans GPU. Pour une utilisation avec CUDA (qu'il faudra installer auparavant), l'url aurait été : https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow-0.9.0rc0-cp35-cp35m-linux_x86_64.whl. Pour les autres configurations (versions de Python / architecture / avec ou sans GPU), vous pourrez consulter la page https://www.tensorflow.org/versions/r0.9/get_started/os_setup.html.

Vous pouvez ensuite tester votre installation via le shell Python :

```
$ python3
Python 3.5.1 (default, Mar 17 2016, 16:05:45)
[GCC 4.9.2] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> import tensorflow
```

2.2 Résolution du problème

Les images de la base d'apprentissage peuvent être récupérée sur le site de la *MNIST database of handwritten digits* :

```
$ mkdir MNIST_data
$ cd MNIST_data
$ wget http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
$ gzip -d train-images-idx3-ubyte.gz
```

Il s'agit d'un format spécifique de données et ne vous attendez pas à pouvoir ouvrir simplement un fichier **.png**. Ce sont des images (55000!) de 20x20 pixels centrées dans des images de 28x28 pixels (*« All images are size normalized to fit in a 20x20 pixel box and there are centered in a 28x28 image using the center of mass. These are important information for our preprocessing »*). Donc si vous voulez travailler sur vos propres fichiers d'image, il faudra les retravailler (centrage, niveau de gris, etc.), mais ceci n'est pas le sujet de cet article.

Il y a trois autres fichiers :

- **train-labels-idx1-ubyte.gz** : ensemble de labels d'apprentissage (indiquant s'il s'agit du chiffre **0**, **1**, **2**, ..., ou **9**) ;
- **t10k-images-idx3-ubyte.gz** : ensemble d'images de test ;
- **t10k-labels-idx1-ubyte.gz** : ensemble de labels de test.

Il existe une façon plus simple d'utiliser ces données : celles-ci sont présentes dans le sous-module **tensorflow.examples.tutorials.mnist**. Nous allons utiliser ces données pour afficher de manière aléatoire l'une des images utilisées dans la base d'apprentissage :

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

EN COUVERTURE

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
01: from tensorflow.examples.tutorials.mnist import input_data
02: import matplotlib.pyplot as plt
03: import matplotlib.cm as cm
04: import matplotlib
05: import random
06: import numpy
07:
08: mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
09: index = random.randint(0, len(mnist.train.images))
10:
11: img = mnist.train.images[index]
12: img = img.reshape((28,28))
13:
14: label = mnist.train.labels[index]
15: num = numpy.where(label == 1)[0][0]
16:
17: fig = plt.figure()
18: fig.suptitle('Écriture manuscrite d\'un {}'.format(num),
fontsize=14, fontweight='bold', color='blue')
19: plt.imshow(img, cmap = cm.Greys)
20: plt.show()
```

Les données sont importées en tant que **input_data** en ligne 1. Les lignes 2 à 6 permettent d'importer les différents modules nécessaires. On lit l'ensemble des données dans la variable **mnist** en ligne 8 puis en ligne 9 on détermine de manière aléatoire un entier faisant référence à un élément de **mnist.train.images** (entier compris entre **0** et la taille de **mnist.train.images**). Dans les lignes 11 et 12 on prend l'image située à l'indice **index** et on la redimensionne en 28x28 pixels. Dans les lignes 14 et 15, on récupère le label associé à l'image sélectionnée. **label** est un **ndarray** de la forme **[0. ... 1. ... 0.]** où le **1.** se situe à l'index correspondant au chiffre représenté (si **label[5] == 1** alors l'image représente un **5**). Comme il s'agit d'un **ndarray**, la recherche s'effectue à l'aide de la fonction **numpy.where()** et renvoie une liste de **ndarray** contenant un seul élément (d'où les **[0][0]**). Enfin, dans les lignes 17 à 20 on affiche l'image en lui associant un titre (ligne 18). La figure 4 (en page suivante) montre le résultat obtenu.

Nous avons fait connaissance avec les données avec lesquelles nous allons travailler, passons maintenant aux choses sérieuses avec la reconnaissance de chiffres manuscrits.

Nous commençons par définir des variables :

```
import tensorflow as tf

e = tf.placeholder(tf.float32, [None, 784])
w = tf.Variable(tf.zeros([784 , 10]))
bias = tf.Variable(tf.zeros([10]))
```

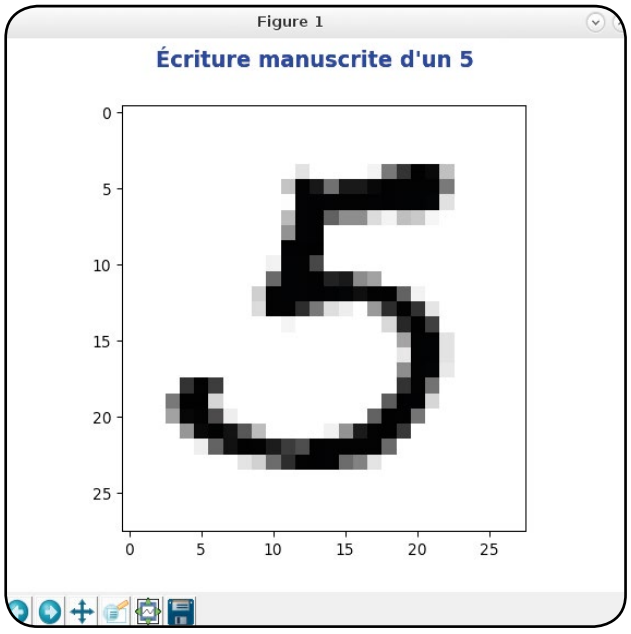


Fig. 4 : Affichage d'un des chiffres de la base d'apprentissage.

e doit pouvoir stocker les données des images de 28x28 pixels (soit **784** pixels). Le poids **w** est associé à chaque pixel avec une probabilité d'être un **0**, un **1**, ..., un **9** (soit **10** possibilités). Enfin le biais est un vecteur ou tenseur d'ordre 1 (d'où le nom de Tensorflow puisque nous manipulons des tenseurs de différents ordres) de 10 zéros.

Pour obtenir une probabilité d'obtention de la reconnaissance d'un chiffre, nous allons utiliser une régression softmax.

La régression est toujours $\sum_{i=1}^{d} w_i e_i + b_i$ et la fonction **softmax()** est telle que $softmax(e)_i = \frac{\exp(e_i)}{\sum_j \exp(e_j)}$ et elle est utilisée en tant que fonction d'activation.

On obtient donc **A** par le code suivant où **tf.matmul()** réalise une multiplication de tenseurs :

```
A = tf.nn.softmax(tf.matmul(e, w) + bias)
```

Nous pouvons ensuite passer à l'étape d'apprentissage. Nous allons utiliser l'entropie croisée **[7]** pour mesurer l'inefficacité (donc l'efficacité) de nos prédictions.

```
y_ = tf.placeholder(tf.float32, [None, 10 ])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(A),
reduction_indices=[1]))
```

Nous minimisons ensuite l'entropie croisée en utilisant un algorithme de descente de gradient (d'autres algorithmes sont disponibles **[8]**) avec un taux d'apprentissage de **0.2** :

```
train_step = tf.train.GradientDescentOptimizer(0.2).
minimize(cross_entropy)
```

Nous initialisons les variables définies :

```
init = tf.initialize_all_variables()
```

Nous créons et démarrons une session Tensorflow :

```
sess = tf.Session()
sess.run(init)
```

Puis nous lançons l'apprentissage (**1000** itérations récupérant un ensemble de **100** données choisies aléatoirement) après avoir lu les données :

```
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

for i in range(1000):
          batch_xs, batch_ys = mnist.train.next_batch(100)
          sess.run(train_step, feed_dict={e: batch_xs, y_:
batch_ys})
```

Nous pouvons enfin vérifier notre taux de bonnes prédictions en utilisant notre modèle sur les données de test (**tf.argmax(A, 1)** est le label prédit et **tf.argmax(y_, 1)** est le label correct) :

```
correct_prediction = tf.equal(tf.argmax(A,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
print(sess.run(accuracy, feed_dict={e: mnist.test.images, y_:
mnist.test.labels}))
```

Ce qui nous permet d'obtenir :

```
$ python3 detect_number.py
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Taux de reconnaissance : 0.91%
```

On peut faire beaucoup mieux **[9]** mais nous nous arrêterons là pour ne pas complexifier inutilement cet article.

Utilisons notre programme pour tenter de reconnaître une des images du jeu de test tirées de manière aléatoire. Ce programme utilisera les notions vues dans les deux programmes précédents.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

ACTUELLEMENT DISPONIBLE
OPEN SILICIUM n°20

COMMENT BIEN CHOISIR VOTRE SYSTÈME DE CONSTRUCTION DE DISTRIBUTION ?
INTRODUCTION À
BUILDROOT & YOCTO

NE LE MANQUEZ PAS
CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :
http://www.ed-diamond.com

EN COUVERTURE

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
01: from tensorflow.examples.tutorials.mnist import input_data
02: import tensorflow as tf
03:
04: import matplotlib.pyplot as plt
05: import matplotlib.cm as cm
06: import matplotlib
07: import random
08: import numpy
09:
10: e = tf.placeholder(tf.float32, [None, 784])
11: w = tf.Variable(tf.zeros([784 , 10]))
12: bias = tf.Variable(tf.zeros([10]))
13:
14: A = tf.nn.softmax(tf.matmul(e, w) + bias)
15:
16: y_ = tf.placeholder(tf.float32, [None, 10 ])
17: cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(A),
reduction_indices=[1]))
18:
19: train_step = tf.train.GradientDescentOptimizer(0.2).
minimize(cross_entropy)
20:
21: init = tf.initialize_all_variables()
22:
23: sess = tf.Session()
24: sess.run(init)
25:
26: mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
27:
28: for i in range(1000):
29:     batch_xs, batch_ys = mnist.train.next_batch(100)
30:     sess.run(train_step, feed_dict={e: batch_xs, y_: batch_
ys})
31:
32: prediction = tf.argmax(A, 1)
33: correct_prediction = tf.equal(prediction, tf.argmax(y_,1))
34: accuracy = tf.reduce_mean(tf.cast(correct_prediction,
"float"))
35: print('Taux de reconnaissance : {:.2f}%'.format(sess.
run(accuracy, feed_dict={e: mnist.test.images, y_: mnist.test.
labels})))
36:
37: index = random.randint(0, len(mnist.test.images))
38:
39: img = mnist.test.images[index]
40: img = img.reshape((28,28))
41:
42: label = mnist.test.labels[index]
43: num = numpy.where(label == 1)[0][0]
44:
45: pred_num = sess.run(prediction, feed_dict={e: mnist.test.
images, y_: mnist.test.labels})[index]
46:
47: fig = plt.figure()
48: fig.suptitle('Écriture manuscrite d\'un {} (prédiction :
{})'.format(num, pred_num), fontsize=14, fontweight='bold',
color='blue')
49: plt.imshow(img, cmap = cm.Greys)
50: plt.show()
```

Le résultat de l'exécution de ce code est visible en figure 5.

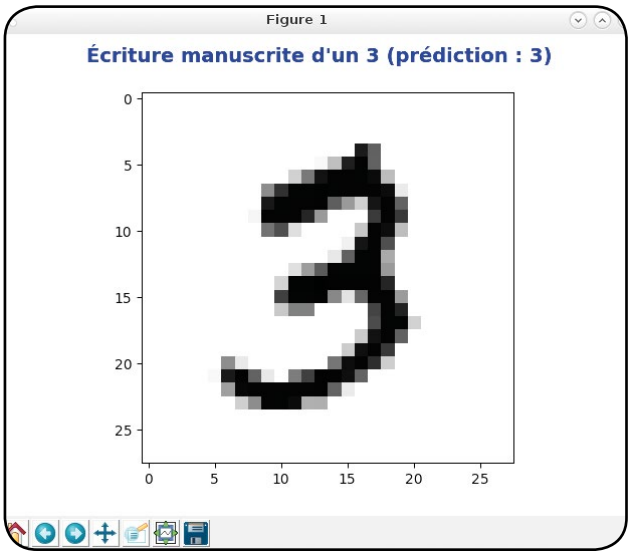


Fig. 5 : Prédiction du chiffre manuscrit d'une image.

2.3 Bonus : affichage de l'impact des poids après apprentissage

En reprenant le code précédant et en en modifiant uniquement la fin, il est possible d'afficher une carte colorimétrique des poids de chaque chiffre :

```
...
06: import matplotlib.pylab as pylab
...
29: for i in range(1000):
30:     batch_xs, batch_ys = mnist.train.next_batch(100)
31:     sess.run(train_step, feed_dict={e: batch_xs, y_: batch_
ys})
32:
33: colormap_dict = {
34:     'red':   [(0.0,  1.0, 1.0),
35:               (0.25, 1.0, 1.0),
36:               (0.5,  0.0, 0.0),
37:               (1.0,  0.0, 0.0)
38:              ],
39:     'green': [(0.0,  0.0, 0.0),
40:               (1.0,  0.0, 0.0)
41:              ],
42:     'blue':  [(0.0,  0.0, 0.0),
43:               (0.5,  0.0, 0.0),
44:               (0.75, 1.0, 1.0),
45:               (1.0,  1.0, 1.0)
46:              ]
47: }
48: redblue = matplotlib.colors.LinearSegmentedColormap('red_
black_blue', colormap_dict, 256)
49:
50: wts = w.eval(sess)
51:
52: for i in range(0,10):
53:     im = wts.flatten()[i::10].reshape((28,-1))
54:     plt.subplot(3, 4, i + 1)
55:     plt.imshow(im, cmap = redblue, clim=(-1.0, 1.0))
```

```
56:     plt.title('Chiffre {}'.format(i), fontsize=14,
fontweight='bold', color='blue')
57:     plt.colorbar()
58:
59: plt.suptitle('Impact des poids', fontsize=14,
fontweight='bold', color='blue')
60: pylab.gcf().canvas.set_window_title('Analyse des poids après
apprentissage')
61: plt.show()
```

La figure 6 montre la carte colorimétrique obtenue pour chaque chiffre : les zones bleues correspondent aux zones dans lesquelles on va chercher un trait et les zones rouges celles dans lesquelles au contraire on ne doit pas avoir de trait pour identifier le chiffre.

Si les couleurs ne vous conviennent pas, vous pouvez modifier la colormap.

À propos de Python 3.5.2

Si vous voulez utiliser la dernière version de Python il va falloir la compiler... mais pas n'importe comment ! En effet, j'en ai fait l'expérience en ayant l'impression d'avoir tout fait correctement mais en n'ayant plus accès à **tkinter**... et donc à **matplotlib** (qui n'affiche aucun message d'erreur ni aucune fenêtre graphique). Si vous avez déjà installé Python 3.5.2 et que le programme proposé en section 2.2 n'affiche rien, testez le code suivant :

```
$ python3
Python 3.5.2 (default, Oct 7 2016, 19:49:49)
[GCC 4.9.2] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> import matplotlib
>>> matplotlib.get_backend()
'agg'
```

Si vous obtenez la sortie **'agg'**, c'est qu'il vous manque le support de **tkinter** ! Pour les autres, les étapes suivantes indiquent comment compiler correctement Python 3.5.2.

1. Assurez-vous tout d'abord de disposer des paquets suivants :

```
$ sudo apt-get install build-essential libreadline-dev
libsqlite3-dev libgdbm-dev libreadline6-dev liblzma-dev libbz2-
dev libncurses5-dev libssl-dev python3-dev tk-dev
```

2. Récupérez et compilez Python 3.5.2 :

```
$ wget https://www.python.org/ftp/python/3.5.2/Python-3.5.2.tar.
xz
$ tar -xvf Python-3.5.2.tar.xz
$ cd Python-3.5.2
$ ./configure
$ make
$ sudo make altinstall
```

Note

L'utilisation de **make altinstall** évite d'écraser le **python3** du système. Si c'est ce que vous souhaitez, alors vous pouvez utiliser **make install**.

3. Créez un environnement virtuel pour Tensorflow et installez-le comme vu précédemment :

```
$ mkvirtualenv tensorflow -p python3.5
(tensorflow) $ export TF_BINARY_URL=https://storage.googleapis.com/
tensorflow/linux/cpu/tensorflow-0.9.0rc0-cp35-cp35m-linux_x86_64.whl
(tensorflow) $ pip install --upgrade $TF_BINARY_URL
(tensorflow) $ git clone https://github.com/matplotlib/matplotlib.
git
(tensorflow) $ cd matplotlib
(tensorflow) $ python setup.py install
```

Note

Pour une installation globale à l'extérieur d'un environnement virtuel utilisez :

```
$ sudo python3 setup.py install
```

4. Re-testez les commandes du départ :

```
$ python3
Python 3.5.2 (default, Oct  7 2016, 19:49:49)
[GCC 4.9.2] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> import matplotlib
>>> matplotlib.get_backend()
'TkAgg'
```

Cette fois-ci vous devez obtenir **'TkAgg'** : tout fonctionne !

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

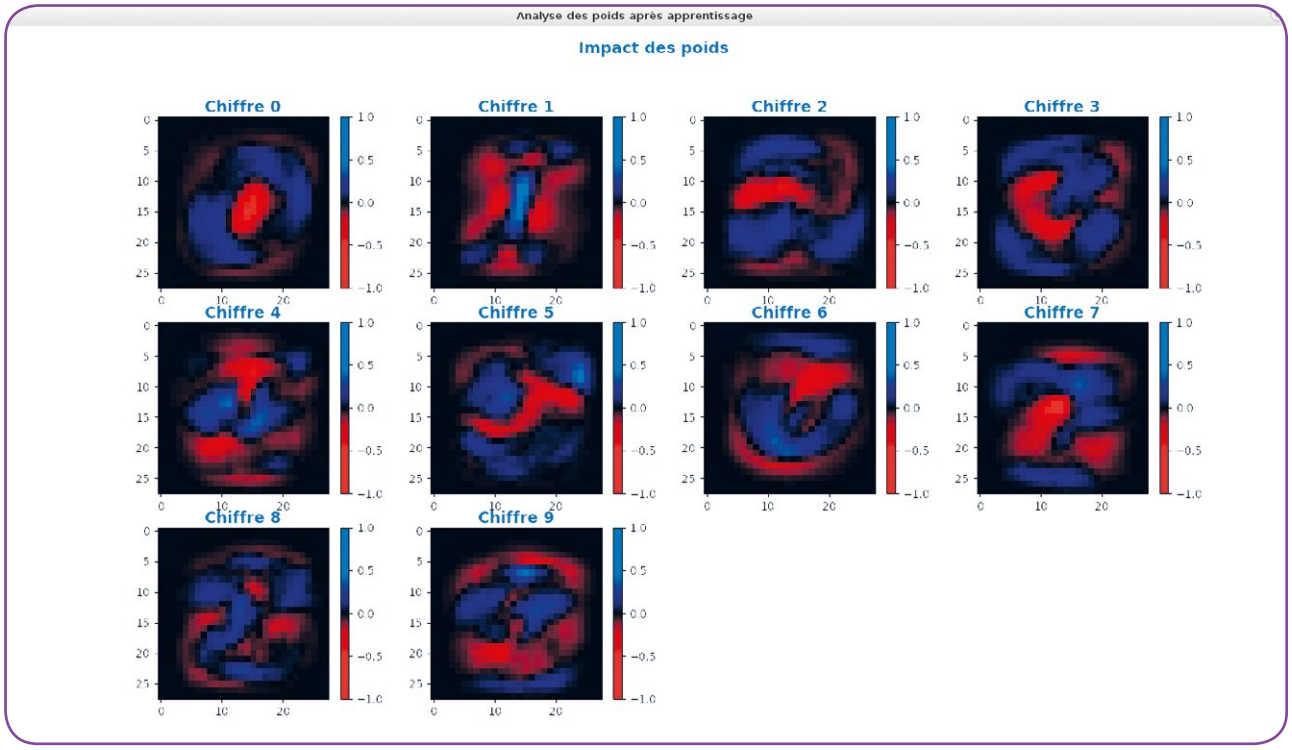


Fig. 6 : Carte colorimétrique des poids de chaque chiffre après apprentissage.

Conclusion

Ainsi s'achève notre tour d'horizon des réseaux de neurones et de Tensorflow. Nous n'avons fait qu'effleurer un domaine particulièrement riche et dense qui a de nombreuses applications dans la prédiction, l'identification et la classification de données. Je suis conscient de n'avoir pas détaillé l'ensemble des notions mathématiques ni d'avoir expliqué pas à pas tout le code utilisé ici, l'objectif étant vraiment de pouvoir commencer à faire quelques pas dans cet univers tout en pouvant y trouver une application concrète. Les portes sont maintenant ouvertes, à vous de poursuivre la route... ■

Remerciements

Un grand merci à Yann Morère et Jean-Baptiste Vioix pour leur relecture et leurs conseils et plus précisément à Yann pour la partie en Scilab dont il est l'auteur.

Références

[1] COLOMBO T., « De la biologie dans du code : les algorithmes génétiques », GNU/Linux Magazine n°171, mai 2014.

[2] ROSENBLATT F., « The Perceptron : A probalistic model for information storage and organization in the brain », Psychological Review, Vol 65(6), novembre 1958, p. 386 à 408.

[3] WIDROW B. et HOFF M. E., « Adaptative switching circuits », IRE WESCON Conv. Rec., 1960, p. 96 à 104.

[4] RUMELHART B., HINTON G. et WILLIAMS R., « Learning representations by back-propagating errors », Nature, 323, 1986.

[5] SOLIS F. et WETS R., « Minimization by random search techniques », Mathematics Of Operations Research, 6, 1981.

[6] Tutoriel de Tensorflow sur la reconnaissance de chiffres manuscrits : https://www.tensorflow.org/versions/r0.9/tutorials/mnist/beginners/index.html

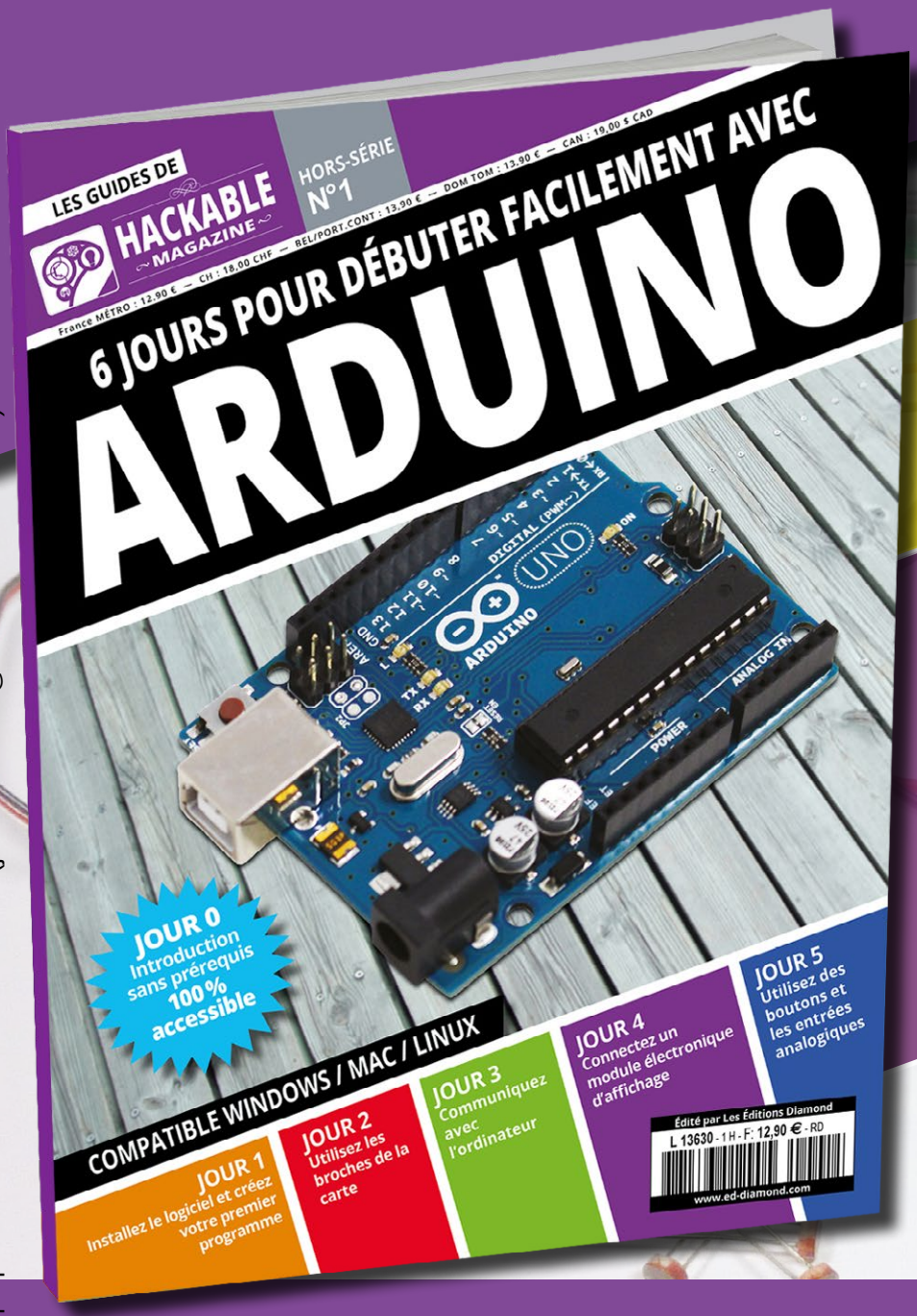
[7] Entropie croisée : https://fr.wikipedia.org/wiki/Entropie_croisée

[8] Algorithmes de calcul de gradients : https://www.tensorflow.org/versions/r0.9/api_docs/python/train.html#optimizers

[9] Amélioration de la prédiction : https://www.tensorflow.org/versions/r0.9/tutorials/mnist/pros/index.html

EN COUVERTURE
Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

VOUS L'AVEZ RATÉ ?
VOUS AVEZ UNE DEUXIÈME CHANCE !



HACKABLE HORS-SÉRIE N°1

VOUS UTILISEZ DÉJÀ LA RASPBERRY PI ? ...OU PAS

METTEZ-VOUS À L'ARDUINO !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)


RÉPARER UN CODE QR

Nicolas PATROIS [Enseignant dans le secondaire]

Nous terminons cette première série sur la lecture des codes QR par la réparation automatique de sa zone de format et de sa zone de données.

Mots-clés : Python, programmation orientée objet, correction d'erreur, Reed-Solomon, algorithme

Résumé

Cet article permet de lire un code QR abîmé ou décoré d'un petit logo en utilisant la correction d'erreur de Reed-Solomon. On corrigera aussi la zone de format.

Dans l'article précédent **[1]**, nous avons codé une bibliothèque qui permet de manipuler le corps $\mathbb{F}_{256}$ et son anneau de polynômes : nous les utilisons pour corriger un code QR défectueux ou muni d'une imagette décorative. Le code est disponible sur GitHub **[2]**.

Deux zones d'un code QR peuvent être corrigées :

- Les zones de format qui contiennent les informations minimales pour commencer à lire les données. Elles sont présentes en double : cinq bits de données et dix de correction.
- La zone centrale qui contient les données découpées en blocs : chacun a ses données brutes et sa correction.

1 Correction de la zone de format

J'ai utilisé les idées de Wikiversity **[3]** pour cette partie.

Voici le véritable code de la fonction qui vérifie la validité des zones de format et qui les corrige si possible.

```
01:   def verifformat(self):
02:     if self.form is None:
03:       self.formats()
04:     r0=bin2dec(self.form[0][5:])
05:     r1=bin2dec(self.form[1][5:])
06:     hamming={i:set() for i in range(16)}
```

On commence par transformer les deux zones de format en nombres, **r0** et **r1** lignes 4 et 5. Ce sont en fait des éléments de $\mathbb{F}_{256}$, c'est pourquoi on utilisera plus loin (lignes 9 et 10) le **XOR** bit à bit.

La table de *hashage* **hamming** contient, en fonction du nombre d'erreurs, les formats possibles. Il suffit ainsi de prendre le minimum des clés pour un format correspondant unique (et non vide) pour obtenir le format correct.

```
07:     for form in range(32):
08:       r=resteformat(form)
09:       reste0=r0^r
10:       reste1=r1^r
11:       c0=bin(reste0).count("1")
12:       c1=bin(reste1).count("1")
13:       hamming[c0].add(form)
14:       hamming[c1].add(form)
```

On teste alors tous les codes possibles, il n'y en a que $2^5=32$, c'est peu donc la force brute est possible et même souhaitable (contrairement à la section suivante).

On calcule le reste de chaque format possible (une division euclidienne où la soustraction est un **XOR** binaire) et où le diviseur est le nombre **1335** soit **10100110111** en binaire. La fonction **resteformat** est dans **qrcodestandard.py**.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
def resteformat(n):
  mod=16*1335 # 0x357
  n*=1024
  for i in range(5):
    if len(bin(n))>=len(bin(mod)):
      n^=mod
    mod//=2
  return n
```

On le compare aux formats présents et on ajoute sa distance de Hamming (le nombre de bits distincts donc le nombre de **1** dans le **XOR** des lignes 9 et 10). Les lignes 13 et 14 ajoutent si nécessaire le format aux formats qui ont la même distance dans **hamming**.

```
15:     m=min(h for h in hamming if
hamming[h])
16:     if m:
17:       self.formatok=False
18:     else:
19:       self.formatok=True
20:     if len(hamming[m])>1:
21:       print("Erreur, conflit de
formats.")
22:       exit(1)
23:     self.form[0]=[int(i) for i in
bin(hamming[m].pop())[2:]]
24:     self.form[0]=[0]*(5-len(self.
form[0]))+self.form[0]
```

Enfin, ligne 15, on cherche la distance de Hamming la plus petite et pour laquelle au moins un format existe. Si elle est nulle, il n'y a pas eu besoin de corriger. Si elle n'est pas nulle, on teste s'il y a plusieurs formats possibles. Si oui, on ne peut pas corriger, il y a ambiguïté. Sinon, on corrige et on remplace.

Fig. 9 : Le code QR à corriger.

2 Correction de la zone de données

Nous allons lire puis corriger le code QR de la figure 9.

Le principe est de transformer la totalité de chaque bloc du message (chaque bloc en clair plus sa correction concaténée) en un polynôme à coefficients dans $\mathbb{F}_{256}$, c'est pourquoi nous avons besoin des deux fonctions **message2poly** et **poly2message** qui font l'interface (dans **qrcorps.py**).

2.1 Interface

Pour corriger un code QR qui contient des erreurs, on a besoin de traduire une liste de bits en un polynôme et inversement.

```
199: def message2poly(message):
200:   poly=[]
201:   for i in range(len(message)//8):
202:     poly.append(F256(bin2dec(message[8*i:8*i+8])))
203:   return Polynome(tuple(poly))
```

Cette fonction transforme une liste de bits en un polynôme en convertissant chaque octet en un élément de $\mathbb{F}_{256}$, la liste des octets donne les coefficients du polynôme.

```
205: def poly2message(poly):
206:   mess=poly.coefficients
207:   liste=[]
208:   for c in mess:
209:     b=[int(i) for i in bin(c.val)[2:]]
210:     liste=liste+[0]*(8-len(b))+b
211:   return liste
```

Cette fonction est la réciproque de la précédente : elle transforme les coefficients d'un polynôme de notre classe en une liste de bits en faisant attention à ce que l'octet ait bien 8 bits.

2.2 Principe mathématique

Le principe est le même, en plus complexe, que la preuve par neuf. Par exemple, on doit transmettre un nombre **n** (par exemple **n = 457859**), on calcule le reste **r** de sa division euclidienne par **9** qui est **2**. Donc le nombre **n × 10 + 9 − r = 4578597** est divisible par **9**. Il suffit de vérifier si le nombre transmis est un multiple de **9** pour savoir s'il y a un problème. Bien évidemment, un message trop altéré ne sera pas détectable, pas plus qu'on ne pourra corriger le message mais l'idée est la même. L'ISBN et le numéro de Sécurité Sociale utilisent ce genre de méthode **[4]**.

Si on se donne un message **m** de **n** octets et l'erreur **e** de **n** octets aussi, le message reçu est **r = m + e** (l'addition n'est pas la concaténation mais l'addition vectorielle). Autrement dit, comme on est en caractéristique **2**, **e = r + m**. Le message **m** est fourni avec sa correction construite de manière à ce que si **a** est notre élément générateur du groupe des inversibles de $\mathbb{F}_{256}$, $\forall x=a^k \in \mathbb{F}_{256}^*, \sum_{j=0}^{n-1} (a^k)^j m_j = 0$

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

où m_0 est le dernier élément du message. Autrement dit si le message est altéré (mais pas trop), cette somme n'est plus nulle.

On utilise le principe de la transformée de Fourier discrète dans les corps finis pour déterminer les positions des erreurs puis leur valeur. Ce n'est pas exactement une transformée de Fourier parce qu'on ne calcule pas la somme sur la totalité des éléments de $\mathbb{F}_{256}$ mais sur une plus petite partie, ici du nombre d'octets correcteurs.

2.3 Le code

La fonction est dans **qrcodestandard.py**, je me suis appuyé sur le travail de B. Barras **[5]**. Il y a plusieurs manières de procéder mais la force brute n'est plus envisageable car si un bloc contient 55 octets soit 440 bits, il faut tester $2^{440} \approx 2{,}839 \times 10^{132}$ cas. J'ai choisi, vu le travail mené précédemment, d'utiliser une version modifiée de la division euclidienne :

```
01: def corrige(clair,redondant):
02:   toutpoly=message2poly(clair+redondant)
03:   syndrome=[toutpoly(F256(F256.exp(i))) for i in
range(len(redondant)//8)]
04:   if set(syndrome)!={F256(0)}:
05:     syndpoly=Polynome(tuple(syndrome[::-1]))
```

Le polynôme **toutpoly** est donc notre polynôme de $\mathbb{F}_{256}[X]$. Ses **n** racines sont les puissances du générateur **2**, de **0** au nombre d'octets moins **1** de la partie redondante, si le message n'est pas entaché d'erreurs. On calcule donc les **n** images par la fonction polynôme associée de ces puissances dans le **syndrome**.

Si toutes les valeurs du **syndrome** sont nulles, le code QR n'est pas endommagé. Remarquer que les éléments de ma classe sont *hashables* sinon on ne pourrait pas créer un ensemble qui en contient.

Si le code est endommagé ligne 04, on transforme le **syndrome** $S = [S_0, S_1, \ldots, S_{n-1}]$ en le polynôme **syndpoly** ligne 05 donc $\text{syndpoly} = \sum_{k=0}^{n-1} S_k X^k$, c'est-à-dire que le coefficient dominant est S_{n-1} et le coefficient constant est S_0.

```
06:     r,v=Polynome.construction([1]+[0]*syndpoly.
degre()),Polynome.construction([0])
07:     rr,vv=syndpoly,Polynome.construction([1])
08:     while r.degre()>=len(syndpoly)//2:
09:       q=r//rr
10:       r,v,rr,vv=rr,vv,r-q*rr,v-q*vv
```

Appelons p_k les positions des erreurs et **v** le polynôme de degré **m** tel que $v = \prod_{k=1}^{m} (1 - a^{p_k} X)$.

L'équation fondamentale est **v × syndpoly + vv × XF = r**, ce qui ressemble fort à une égalité de Bezout. Il faut cependant arrêter le calcul avant que le dernier reste ne soit nul et recoder l'algorithme.

```
11:     vder=v.der()
12:     racines=[i for i in range(255,255-len(toutpoly),-1) if
v(F256(F256.exp(i)))==F256(0)]
```

On détermine la dérivée de **v** (qui nous servira pour déterminer les erreurs) puis ses racines b_k, directement reliées aux positions des erreurs (ce sont les inverses, $b_k = (a^{p_k})^{-1}$).

```
13:     if 2*len(racines)>=len(syndrome):
14:       print("Il y a trop d'erreurs dans le bloc
n°"+str(ii)+".",file=sys.stderr)
15:       exit(1)
```

S'il y a trop d'erreurs, on précise le bloc concerné et on sort.

```
16:     erreurs={i:r(F256(F256.exp(i)))/vder(F256(F256.exp(i)))/
F256(F256.exp(i)) for i in racines}
```

On calcule les valeurs des erreurs pour chaque racine b_k de **v**, c'est-à-dire $\frac{r(b_k)}{v'(b_k)}$.

```
17:     for i in erreurs:
18:       toutpoly[255-i]+=erreurs[i]
19:     toutmessage=poly2message(toutpoly)
20:     return toutmessage[:len(clair)],toutmessage[len(clair):]
,len(racines)
21:   return clair,redondant,0
```

On corrige les erreurs du bloc courant et on retourne le bloc en clair corrigé, le bloc redondant corrigé et le nombre d'erreurs corrigées.

Si on n'a pas demandé de corriger, on retourne tout tel quel et **0** (aucune erreur corrigée).

```
> ./qrdecode.py -i linux.png -a1
Fichier : linux.png
Niveau de correction : Low
Masque :
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
Dimensions : 105×105
Version : 22
Il y a eu besoin de corriger 61 erreur(s) dans la zone de données.
Mode : Byte
Longueur du message : 949
Message :
Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big
and
...
```

Bon, je coupe, vous avez compris la teneur de la suite du message.

Conclusion

La lecture des codes QR est terminée. On aurait pu coder d'autres algorithmes de correction par curiosité.

Maintenant, il nous reste à créer nos propres codes QR... ■

Références

[1] PATROIS N., « Fabriquer un corps fini », GNU/Linux Magazine n°196, septembre 2016, p. 32 à 37.

[2] Le code et les exemples sont présents ici : https://github.com/GLMF/GLMF198

[3] Reed–Solomon codes for coders : https://en.wikiversity.org/wiki/Reed%E2%80%93Solomon_codes_for_coders.

[4] EVEILLEAU T., « Les clés de détection d'erreur » : http://therese.eveilleau.pagesperso-orange.fr/pages/truc_mat/textes/cles.htm.

[5] BARRAS B., « Codes de Reed Solomon », 1999 : http://infoscience.epfl.ch/record/198834/files/rs.pdf?version=1.

Enseignants, Lycées, Écoles, Universités...

LINUX MAGAZINE / FRANCE

C'est possible ! Rendez-vous sur :

http://proboutique.ed-diamond.com

pour consulter les offres !

N'hésitez pas à nous contacter pour un devis personnalisé par e-mail : abopro@ed-diamond.com ou par téléphone : +33 (0)3 67 10 00 20

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

TRIER AUTOMATIQUEMENT LES FICHIERS D'UN RÉPERTOIRE AU FIL DE LEUR APPARITION

Tristan COLOMBO

La « téléchargite » est une maladie consistant à télécharger de nombreux fichiers sur le Web. Ces fichiers viennent se placer dans le répertoire de téléchargement qui devient rapidement une seconde corbeille... Pour mettre un peu d'ordre dans ce répertoire, je vous propose d'implémenter un mécanisme qui triera automatiquement les fichiers qui seront placés dans ce répertoire en fonction de leur extension.

inotify incrontab incron automatisation

L'OBJECTIF

Un répertoire donné sert de dépôt à un logiciel qui vient y stocker des fichiers (typiquement le répertoire de téléchargement d'un navigateur Web). Ce répertoire devient rapidement une « poubelle » dans laquelle on ne peut plus rien retrouver (en tout cas chez moi c'est le cas...). Nous allons mettre en place un mécanisme qui triera pour nous les nouveaux fichiers en fonction de leur extension. Ainsi les pdf, les images, etc. auront chacun leur répertoire distinct.

LES OUTILS

- un système GNU/Linux avec un noyau >= 2.6.13 (`uname -a` pour connaître la version de votre noyau) ;
- un shell bash ;
- un éditeur de code (`vim` par exemple).

PHASE 1

Installer incron

Inotify est un mécanisme présent depuis le noyau 2.6.13 qui permet de transmettre des notifications en surveillant le système de fichiers (création, modification, suppression, etc.). Les principaux événements pouvant être surveillés sont résumés dans le tableau suivant (voir `man inotify` pour une description complète).

Pour pouvoir détecter un événement et y associer le déclenchement d'une action nous utiliserons `incron` (*INotify CRONtab*) dont le fonctionnement est très proche du `cron` traditionnel. Pour y avoir accès, il faut bien entendu commencer par l'installer :

```
$ sudo apt install incron
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Événement	Description
IN_ACCESS	Accès au fichier en lecture.
IN_ATTRIB	Modification des métadonnées (permissions, horodatages, etc.).
IN_CLOSE_WRITE	Fermeture d'un fichier après écriture.
IN_CREATE	Création d'un fichier (n'attend pas la fermeture de celui-ci pour déclencher une action).
IN_DELETE	Suppression d'un fichier.
IN_MOVED_FROM	Déplacement d'un fichier.

Le fichier **/etc/incron.allow** définit la liste des utilisateurs ayant le droit d'utiliser incron. Ici nous allons ajouter l'utilisateur courant (votre *login*) en éditant ce fichier en tant que root :

```
login_utilisateur_courant
```

Pour appliquer toute modification à incron il faut redémarrer le démon **incrond** :

```
$ sudo service incron restart
```

PHASE 2

Détecter l'apparition d'un fichier

Les règles de surveillance sont spécifiées à l'aide de la commande **incrontab** qui permet de les ajouter simplement à la manière d'un **crontab**. Toute règle doit être de la forme :

```
<répertoire_à_surveiller> <événement> <action>
```

Pour commencer, nous allons surveiller l'apparition d'un nouveau fichier dans **~/Telechargements** (qu'il faudra indiquer en chemin absolu) et afficher une petite fenêtre signalant l'apparition du fichier dans le répertoire. De manière à utiliser le nom des éléments liés à un événement venant d'être détecté, nous avons accès aux variables suivantes :

Variable	Description
$#	Nom du fichier/dossier ayant provoqué l'événement.
$@	Nom du répertoire dans lequel l'événement a été détecté.
$%	Nom de l'événement ayant déclenché l'action.

Pour ajouter notre règle nous allons donc appeler **incrontab** :

```
$ incrontab -e
/home/login_utilisateur_courant/
Telechargements IN_CREATE /home/login_
utilisateur_courant/bin/exec_incron $#
```

exec_incron est un script que nous avons placé dans **~/bin/exec_incron** :

```
01: #!/bin/bash
02:
03: usage() {
04:   echo "$(basename ${0}) <filename>"
05:   echo "  Display a notification
about the file <filename>"
06: }
07:
08: if [ ${#} -ne 1 ]; then
09:   usage
10:   exit 1
11: fi
12:
13: export DISPLAY=:0.0
14: /usr/bin/notify-send "Apparition du
fichier ${1}"
```

Il faut absolument penser à exporter la variable **DISPLAY** pour accéder à l'affichage graphique requis par la commande **notify-send** de la ligne 14. Le **${1}** de cette ligne 14 correspond au **$#** transmis dans notre règle incrontab. Rendez ce script exécutable :

```
$ chmod +x ~/bin/exec_incron
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Pensez ensuite à relancer **incrond** :

```
$ sudo service incron restart
```

Pour tester notre script, nous n'avons plus qu'à nous rendre dans le répertoire **~/Telechargements** et à y créer un fichier :

```
$ cd ~/Telechargements
$ touch nouveau.txt
```

En haut et à droite de votre bureau, vous devriez voir apparaître une fenêtre semblable à celle présentée en figure 1.

Fig. 1 : Notification de création du fichier ~/Telechargements/nouveau.txt

PHASE 3

Déplacer les nouveaux fichiers dans un répertoire dédié

Nous avons compris le mécanisme d'incron et nous pouvons donc l'appliquer pour résoudre notre problème. Pour cela il nous faut un script prenant un nom de fichier en paramètre : s'il s'agit d'un répertoire, on ne fait rien, sinon, en fonction de l'extension on déplace le fichier. Par exemple, les fichiers d'extension **.jpg**, **.png** et **.gif** seront placés dans **images**. Créons donc un fichier **~/bin/exec_incron** :

```
01: #!/bin/bash
02:
03: EXT_IMAGES=(jpg png gif)
04: EXT_AUDIO=(mp3 wav)
05: EXT_VIDEO=(mkv avi mp4)
06: EXT_DOC=(odt doc pdf txt)
07: EXT_ARCHIVES=(tgz tar zip gz deb rpm)
08: EXT_ALL=(EXT_IMAGES EXT_AUDIO EXT_VIDEO EXT_DOC EXT_ARCHIVES)
09:
10: usage() {
11:   echo "$(basename ${0}) <filename>"
12:   echo "  Move the file <filename> in a dedicated directory"
13: }
14:
15: move_in() {
16:   file=${1}
17:   dir=${2}
18:
19:   if ! [ -d ~/Telechargements/${dir} ]; then
20:     mkdir ~/Telechargements/${dir}
21:     /usr/bin/notify-send "Telechargements" \
22:                          "Création de ${dir}"
23:   fi
24:
25:   mv ~/Telechargements/${file} ~/Telechargements/${dir}
26:   if [ ${?} -eq 0 ]; then
27:     /usr/bin/notify-send "Telechargements (${dir})" \
28:                          "Déplacement de ${file}"
29:   fi
30: }
31:
32: move_file() {
33:   if [ -d ~/Telechargements/${1} ]; then
34:     /usr/bin/notify-send "Telechargements" \
35:                          "${1} est un repertoire
36: - Aucune action -"
37:   else
38:     ext="${1##*.}"
39:     ext="${ext,,}"
40:     for ext_dir in ${EXT_ALL[@]}; do
41:       write_dir="${ext_dir,,}"
42:       write_dir="${write_dir:4}"
43:       eval ext_current=\( \${${ext_dir}[@]} \)
44:       if [[ ${ext_current[@]} =~ ${ext} ]]; then
45:         move_in ${1} ${write_dir}
46:       fi
47:     done
48:   fi
49: }
50:
51: run() {
52:   if [ $# -ne 1 ]; then
53:     usage
54:     exit 1
55:   fi
56:
57:   export DISPLAY=:0.0
58:   move_file ${1}
59: }
60:
61:
62: run "${@}"
```

Ce script nécessite quelques explications. Dans les lignes 3 à 8 se trouvent les variables contenant les listes d'extensions pour lesquelles les fichiers seront placés dans un même répertoire. Le nom de ces variables servira à déterminer le nom du répertoire de stockage en les passant en minuscules et en supprimant les quatre premiers caractères. Ainsi, les extensions **jpg**, **png** ou **gif** se trouvant dans le tableau **EXT_IMAGES**, les fichiers portant cette extension seront déplacés dans le répertoire **images**.

En ligne 8 le tableau **EXT_ALL** contient le nom de toutes les variables que nous utiliserons pour classer les fichiers (listes d'extensions).

Dans les lignes 10 à 13 nous définissons la fonction **usage()** indiquant comment utiliser le script.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Dans les lignes 15 à 30 nous définissons la fonction **move_in()** qui accepte deux paramètres : le nom d'un fichier et le nom d'un répertoire (ces paramètres sont récupérés en ligne 16 et 17 dans **${file}** et **${dir}** pour une meilleure lisibilité). Ensuite, dans les lignes 19 à 23, si le répertoire **~/Téléchargements/${dir}** n'existe pas alors il est créé. Puis, en ligne 25 le fichier est déplacé et les lignes 26 à 29 permettent d'afficher une notification graphique si le déplacement a bien eu lieu (**${?}** est le statut de retour de la dernière commande exécutée - ici **mv** ; **0** indique que tout s'est déroulé correctement).

Dans les lignes 32 à 49 nous définissons la fonction **move_file()** qui prend en paramètre le nom du fichier à déplacer. Si le fichier est un répertoire, on ne fait rien et on affiche un petit message pour le signaler (lignes 33 à 36). Si ce n'est pas un répertoire, nous récupérons son extension (les caractères se situant après le dernier point dans la variable **${1}** comme l'indique l'expression **${1##*.}** de la ligne 38). En ligne 39 nous passons la variable **${ext}** en caractères minuscules (par **${ext,,}**) pour ne pas être sensible à la casse. Nous parcourons ensuite le tableau **${EXT_ALL}** et pour chaque nom de variable (**EXT_IMAGES**, **EXT_AUDIO**, etc.) que nous récupérons dans **${ext_dir}**, nous passons ce nom en minuscules (ligne 41) et nous supprimons ses quatre premiers caractères par **${write_dir:4}** (ligne 42). Cela nous donne le nom du répertoire de destination (pour **EXT_IMAGES** nous obtenons **images**, etc.). En ligne 43 nous voulons associer à la variable **${ext_current}** le contenu du tableau dont le nom est stocké dans la variable **${ext_dir}** et on utilise pour cela la fonction **eval**. Enfin, si l'extension du fichier en cours est présente dans la liste d'extensions du tableau, on appelle la fonction **move_in** pour déplacer le fichier dans le bon répertoire.

La fonction **run()** des lignes 51 à 59 teste simplement si le script est correctement appelé (lignes 52 à 55), définit la variable d'environnement **DISPLAY** pour l'affichage graphique (ligne 57) et appelle **move_file** qui effectue tout le travail (ligne 58). En ligne 62 le script débute réellement en appelant justement la fonction **run** et en lui transmettant les paramètres reçus.

Vous devrez ensuite modifier votre règle incrontab de manière à ne déplacer un fichier que lorsque celui-ci a été entièrement téléchargé **IN_CLOSE_WRITE** à la place de **IN_CREATE** :

```
$ incrontab -e
/home/login_utilisateur_courant/Telechargements IN_CLOSE_WRITE
/home/login_utilisateur_courant/bin/exec_incron $#
```

Relancez ensuite **incrond** :

```
$ sudo service incron restart
```

C'est prêt ! Téléchargez des fichiers sur le Web, votre répertoire **~/Telechargements** sera toujours classé...

PHASE 4

Pour les utilisateurs de Firefox ou Iceweasel

Le script présenté dans cet article ne fonctionnera pas avec les navigateurs basés sur Firefox. En effet, ces navigateurs commencent par créer un fichier **fichier.ext**, téléchargent le fichier dans **fichier.ext.part**, puis renomment **fichier.ext.part** en **fichier.ext**. Il faut donc modifier légèrement le script et changer l'événement intercepté par incron sous peine de déplacer un fichier vide.

Les modifications sur **exec_incron** sont les suivantes et seront portées dans un nouveau fichier **exec_incron_part** :

```
...
32: move_file() {
33:   if [ -d ~/Telechargements/${1} ]; then
34:     /usr/bin/notify-send "Telechargements" \
35:                          "${1} est un repertoire
36: - Aucune action -"
37:   else
38:     ext="${1##*.}"
39:     ext="${ext,,}"
40:     if [ ${ext} == "part" ]; then
41:       ext_file="${1%.*}"
42:       ext_file="${ext_file##*.}"
43:       for ext_dir in ${EXT_ALL[@]}; do
44:         write_dir="${ext_dir,,}"
45:         write_dir="${write_dir:4}"
46:         eval ext_current=\( \${${ext_dir}[@]} \)
47:         if [[ ${ext_current[@]} =~ ${ext_file} ]]; then
48:           move_in ${1%.*} ${write_dir}
49:         fi
50:       done
51:     fi
52:   fi
53: }
...
```

L'expression **${1%.*}** permet de récupérer le nom du fichier avant la dernière extension (on supprime **.part**).

Vous vous demandez sans doute pourquoi nous n'avons pas simplement modifié le script **exec_incron** : j'ai gardé le meilleur pour la fin ! Firefox ne télécharge pas toujours

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

les fichiers en ajoutant l'extension **.part**, parfois c'est un téléchargement « direct » et il faut donc conserver l'ossature du script **exec_incron** en indiquant de ne traiter que les fichiers ne possédant pas l'extension **.part** :

```
...
32: move_file() {
33:   if [ -d ~/Telechargements/${1} ]; then
34:     /usr/bin/notify-send "Telechargements" \
35:                          "${1} est un repertoire
36: - Aucune action -"
37:   else
38:     ext="${1##*.}"
39:     ext="${ext,,}"
40:     if [ ${ext} != "part" ]; then
41:       for ext_dir in ${EXT_ALL[@]}; do
42:         write_dir="${ext_dir,,}"
43:         write_dir="${write_dir:4}"
44:         eval ext_current=\( \${${ext_dir}[@]} \)
45:         if [[ ${ext_current[@]} =~ ${ext} ]]; then
46:           move_in ${1} ${write_dir}
47:         fi
48:       done
49:     fi
50:   fi
51: }
...
```

Ne pas oublier ensuite de modifier incrontab :

```
$ incrontab -e
/home/login_utilisateur_courant/Telechargements IN_CLOSE_WRITE
/home/login_utilisateur_courant/bin/exec_incron $#
/home/login_utilisateur_courant/Telechargements IN_MOVED_FROM
/home/login_utilisateur_courant/bin/exec_incron_part $#
```

Relancez le démon **incrond** et... ce qui paraissait être une solution ne l'est pas vraiment :

- incron intercepte tous les événements d'un répertoire pour déclencher une action : ici seul l'événement **IN_CLOSE_WRITE** sera traité. Il faut appeler un seul script auquel on transmet le type d'événement (**$%**) ;
- il faudrait maintenir deux scripts et donc effectuer tout ajout de prise en charge d'une nouvelle extension en double : le risque d'erreur est trop important !

PHASE 5

Un script pour les gouverner tous !

Finalement la solution à notre problème sera un script capable de gérer les fichiers « standards » sur un événement **IN_CLOSE_WRITE** et les fichiers partiels **.part** sur un événement **IN_MOVED_FROM**. Voici ce nouveau script dans lequel la fonction **move_in()** reste inchangée :

```
01: #!/bin/bash
02:
03: EXT_IMAGES=(jpg png gif)
04: EXT_AUDIO=(mp3 wav)
05: EXT_VIDEO=(mkv avi mp4)
06: EXT_DOC=(odt doc pdf txt)
07: EXT_ARCHIVES=(tgz tar zip gz deb rpm)
08: EXT_ALL=(EXT_IMAGES EXT_AUDIO EXT_VIDEO EXT_DOC EXT_ARCHIVES)
09:
10: usage() {
11:   echo "$(basename ${0}) <type> <filename>"
12:   echo "  <type> : IN_CLOSE_WRITE - standard file"
13:   echo "           IN_MOVED_FROM - part file"
14:   echo ""
15:   echo "  Move the file <filename> in a dedicated directory"
16: }
17:
18: move_in() {
...
33: }
34:
35: move_file() {
36:   if [ -d ~/Telechargements/${2} ]; then
37:     /usr/bin/notify-send "Telechargements" \
38:                          "${2} est un repertoire
39: - Aucune action -"
40:   else
41:     ext="${2##*.}"
42:     ext="${ext,,}"
43:     if [[ ${1} == "IN_MOVED_FROM" && ${ext} == "part" ]]; then
44:       ext="${2%.*}"
45:       ext="${ext##*.}"
46:     fi
47:     for ext_dir in ${EXT_ALL[@]}; do
48:       write_dir="${ext_dir,,}"
49:       write_dir="${write_dir:4}"
50:       eval ext_current=\( \${${ext_dir}[@]} \)
51:       if [[ ${ext_current[@]} =~ ${ext} ]]; then
52:         if [ ${1} == "IN_CLOSE_WRITE" ]; then
53:           move_in ${2} ${write_dir}
54:         else
55:           move_in ${2%.*} ${write_dir}
56:         fi
57:       fi
58:     done
59:   fi
60: }
61:
62: run() {
63:   if [ ${#} -ne 2 ]; then
64:     usage
65:     exit 1
66:   fi
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
67:
68:   export DISPLAY=:0.0
69:   move_file ${1} ${2}
70: }
71:
72: run "${@}"
```

Le premier paramètre de ce script sera le fameux événement **$%** qui aura donc pour valeur **IN_CLOSE_WRITE** ou **IN_MOVED_FROM**. Pour le reste, il s'agit d'une compilation des scripts **exec_incron** et **exec_incron_part** précédents.

Il n'y aura plus qu'une seule règle incrontab :

```
$ incrontab -e
/home/login_utilisateur_courant/Telechargements IN_CLOSE_WRITE,IN_
MOVED_FROM /home/login_utilisateur_courant/bin/exec_incron $% $#
```

Cette fois-ci après redémarrage du démon, tout fonctionnera suivant notre cahier des charges de départ.

Note

Lors du téléchargement d'un fichier partiel sous Firefox nous obtiendrons deux affichages indiquant le déplacement du fichier. Cela est dû au fonctionnement de Firefox qui crée le fichier **fichier.ext** (événement **IN_CLOSE_WRITE** qui déclenche le déplacement du fichier vide), crée le fichier **fichier.ext.part** (extension non reconnue), puis déplace **fichier.ext.part** dans **fichier.ext** (événement **IN_MOVED_FROM** qui déclenche le déplacement du fichier et l'écrasement du fichier vide).

La seule façon de modifier ce comportement serait de modifier la manière dont Firefox télécharge les fichiers. Certaines extensions telles que **SaveFileTo** le permettent (voir https://addons.mozilla.org/en-US/firefox/addon/save-file-to/).

LE RÉSULTAT

Nous avons obtenu un système automatisé qui va déplacer et classer les fichiers téléchargés (ou copiés) dans le répertoire **~/Telechargements**. Notez toutefois une limitation de ce script : il ne fonctionne pas avec les noms de fichiers contenant des espaces (ce qui ne devrait pas exister, mais sur internet on ne choisit pas le nom des fichiers). Vous pouvez donc améliorer le script pour qu'il soit capable de traiter ces fichiers... ou être optimiste et considérer comme moi que les gens nomment correctement leurs fichiers la plupart du temps. ■

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

LINAGORA SOUTIENT

Linux Professional Institute

la vraie CERTIFICATION INDÉPENDANTE

Maximisez vos chances de réussite !

(taux de satisfaction 95%)

NOS SESSIONS POUR DÉCEMBRE 2016

PARIS
- LPI 102 — 5 au 8
- EXAMEN LPI — 8

TOULOUSE
- LPI 201 — 5 au 8
- LPI 202 — 12 au 15
- EXAMEN LPI — 15

formation.LINAGORA.com

LES RÉSEAUX LOGIQUES (VLANS)

Nicolas GRENECHE [Ingénieur de recherche à l'Université Paris 13]
Youssef ABDECHCHAFIQ [Ingénieur d'étude à l'université Paris 13]

Avant l'avènement des VLANs, il était impératif d'être physiquement présent sur le backbone d'un réseau pour faire partie du même domaine de broadcast. Cette contrainte limitait considérablement les possibilités de déplacement géographique de collaborateurs voulant travailler sur le même réseau tout en occupant des bureaux physiquement distants de plusieurs bâtiments. Vu les coûts induits par la multiplication de liaisons et l'acquisition de matériel de routage pour chaque réseau, cette technologie permettant de se soustraire à cette contrainte physique et financière est incontournable pour les architectes réseau.

Mots-clés : VLAN, 802.1Q, OSI, Routage, Topologie, Linux

Résumé

Vous trouverez dans cet article les notions de base sur les VLANs, leur intégration dans une infrastructure réseau et des exemples concrets de configuration sur un système GNU/Linux.

1 Un peu de théorie

Pour bien comprendre ce que sont les VLANs (*Virtual Local Area Network*), il convient d'introduire certains concepts réseau théoriques. Pas de panique, la pratique sera extrapolée vers la théorie et non l'inverse. Nous allons commencer par discuter les modèles OSI et TCP/IP via le scénario d'un utilisateur effectuant une connexion SSH sur une machine distante située sur le même commutateur (ou *switch*). Cette introduction au modèle OSI nous aidera à positionner les VLANs dans la pile réseau. Nous allons ensuite préciser le phénomène d'encapsulation. C'est primordial pour l'étude des VLANs car toute la logique repose sur ce phénomène.

1.1 Les modèles OSI et TCP/IP

Quand on parle de réseau, une foultitude de termes vient immédiatement à l'esprit : RJ45, WiFi, fibre optique, 802.3, Ethernet, TCP/IP, etc. Ces termes se réfèrent soient à des médias, soit à des protocoles. Une communication réseau est un transfert d'information entre deux pairs. Le média est le canal physique utilisé par les deux pairs pour communiquer. Le protocole structure l'information de la communication afin qu'elle soit compréhensible par les deux pairs. Une communication peut impliquer une succession de médias hétérogènes. Par exemple, lorsque vous vous connectez à un site Web distant, le premier média peut être votre réseau sans fil entre votre ordinateur et votre box. La communication passe ensuite en ADSL sur Internet. Elle termine ensuite en filaire chez

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

le propriétaire du site Web. La même information transite donc sur trois médias différents. On touche du doigt le fait qu'il doit y avoir une indépendance totale entre le média et la communication réseau à proprement parler.

Pour illustrer le modèle OSI, prenons l'exemple d'un utilisateur sur une machine **A** se connectant en SSH à une autre machine **B** située sur le même segment LAN. Dans notre exemple, le segment LAN est l'ensemble des machines connectées au commutateur (ou *switch*). Les deux renseignements que l'on fournit à la commande SSH sont le nom d'utilisateur distant et l'IP de la machine ciblée. Le port TCP du serveur SSH destination est implicitement fixée à **22**. Nous avons donc trois protocoles explicitement impliqués : IP, TCP et SSH. IP fait transiter les données entre les deux machines. TCP établit la connexion entre les deux extrémités concernées (qui sont en fait des sockets réseaux) à savoir le client SSH sur la machine **A** et le serveur SSH sur la machine **B**. En effet, chaque machine peut héberger de multiples services réseau et aussi établir des connexions vers plusieurs destinations. Il ne suffit donc pas d'être capable d'acheminer l'information d'un point **A** à un point **B** (IP), il faut être capable de savoir à quel service de **B** livrer ces informations (TCP). On peut dire qu'IP et TCP sont complémentaires (c'est pour ça qu'on parle de TCP/IP, TCP sur IP), ou plus précisément que TCP utilise les services d'IP pour établir une communication de bout en bout entre deux extrémités (socket) réseau. Une extrémité est donc un couple IP / Port dans le monde TCP/IP. Enfin, le protocole SSH définit les primitives utilisées par le processus client et le processus serveur SSH pour communiquer entre eux via la connexion TCP/IP établie.

Il manque une pièce dans le puzzle présenté ci-dessus. Nous n'avons considéré que des extrémités « logiques ». À un moment, il faut quand même acheminer les paquets réseaux d'une interface physique de **A** à l'autre sur **B**. Ceci est possible grâce à l'utilisation d'un protocole chargé de gérer le média physique (c'est-à-dire les interfaces et le câble) : Ethernet 802.3. Ce protocole attache à chaque interface une adresse MAC (*Media Access Control*). Cette adresse est inscrite dans le contrôleur de l'interface, elle n'est pas configurée dans le système d'exploitation (bien que les outils du système d'exploitation puissent la changer). En résumé, nous avons donc une communication SSH qui dépend de TCP (pour toucher la bonne extrémité sur la machine cible), s'appuyant sur IP (pour toucher la bonne interface logique) et enfin le tout utilise l'Ethernet 802.3 (pour toucher la bonne interface physique).Un modèle académique a été introduit pour normaliser ces différentes couches. Il s'agit du modèle OSI (*Open Systems Interconnection*). Ce modèle se compose de 7 couches, de la plus basse à la plus haute : physique, liaison, réseau, transport, session, présentation et application.

La couche physique concerne la transmission du signal. Il s'agit de transmettre les bits de données de **A** vers **B**. La couche liaison (Ethernet 802.3) gère la communication de deux machines reliées par un même média physique (cas de deux machines interconnectées par un commutateur). La couche réseau (IP) intègre des mécanismes de routage pour livrer des paquets qui ne sont pas forcément à destination du segment LAN. La couche transport (TCP ou UDP) définit des ports qui sont attachés (*bind*) à un service réseau. La couche session gère les transactions, essentiellement les ouvertures et fermetures de session. La couche présentation définit le codage des données applicatives transmises. Enfin la couche application contient toute la batterie de protocoles de haut niveau utilisée par les applications des utilisateurs finaux (HTTP, SSH, DNS etc.).

Note

Communications IP sur LAN

D'après le dernier paragraphe, on pourrait penser que lorsque deux machines communiquent sur le même segment LAN elles pourraient se passer de la couche réseau. En théorie oui car le niveau liaison suffit pour établir un canal de communication entre deux machines du même segment LAN. En pratique, la couche réseau est tout de même utilisée dans un souci de traitement uniforme des trames reçues par la pile réseau.

1.2 En-têtes et encapsulation

Dans un modèle en couches, l'idée est que la couche **N** va utiliser les services de la couche **N-1**. Chaque protocole fonctionne sur le modèle en-tête suivi de la charge applicative (*payload*). Reprenons le modèle OSI à partir de la couche liaison. Nous avons vu que le service au niveau de cette couche est assuré par le protocole Ethernet 802.3. Ce protocole prend en en-tête l'adresse MAC destination et source. Sa charge applicative est le paquet IP. IP est donc encapsulé dans la trame Ethernet 802.3. La couche réseau utilise les services de la couche liaison. L'en-tête d'un paquet IP est l'adresse destination suivi de la source. Sa charge applicative est le paquet TCP. L'en-tête du paquet TCP contient les ports source et destination. Sa charge applicative est la requête SSH. Ces encapsulations sont à la charge de la

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

pile réseau de l'émetteur. Ces encapsulations successives sont schématisées dans la figure 1. Le décapsulage est à la charge du destinataire.

Fig. 1 : Encapsulation

1.3 De l'IP vers l'Ethernet

Reprenons l'exemple de la connexion SSH et plus précisément l'envoi d'un paquet réseau de la machine **A** vers la machine **B**. Tout ce qui est au-dessus du niveau 3 (réseau) du modèle OSI est hors sujet de cet article. Le propos ici est d'amener un paquet de l'interface réseau de **A** jusqu'à l'interface de **B**. Au niveau de **A**, nous ne connaissons que l'IP de **B**. Comme nous l'avons vu dans la section précédente, chaque interface réseau dispose de sa propre adresse MAC (adresse « physique », niveau 2 de l'OSI). **A** va donc commencer par récupérer l'adresse MAC de **B**. Cela se fait via un protocole nommé ARP (*Address Resolution Protocol*). Une adresse MAC à une taille de 6 octets et elle est souvent présentée en hexadécimal. **A** va envoyer une requête ARP *who-has* à une adresse MAC particulière, l'adresse de *broadcast* (**FF:FF:FF:FF:FF:FF**). Le principe du *broadcast* est d'arroser « tout le réseau ». Cette notion de « tout le réseau » est différente selon le niveau du modèle OSI où on se positionne. Si on est au niveau liaison en Ethernet, il s'agit du segment LAN. Si on est au niveau réseau en IP, il s'agit de toute la plage. Cette requête sert à demander « quel est l'adresse MAC associée à l'IP B ? ». Comme c'est envoyé sur l'adresse de *broadcast*, toutes les machines du segment LAN reçoivent cette requête. La machine concernée (**B**) envoie une réponse ARP reply à **A** avec son adresse MAC. À partir de là **A** peut établir une liaison (au sens niveau 2 du modèle OSI) avec **B**.

2 Les VLANs

Les VLANs sont définis dans la norme 802.1Q. Cette norme spécifie un tag qui est inséré dans les en-têtes de la trame Ethernet. Ce tag comprend essentiellement un VID (*VLAN Identifier*). En effet chaque VLAN est défini par un entier positif, le VID. Ce tag peut être ajouté au moment de la génération de la trame par la machine ou par le commutateur lors du passage de la trame (voir figure 2). Ce dernier cas entraîne un recalcul du FCS (*Frame Check Sequence*). L'idée est que ce tag indique au commutateur à quel VLAN circonscrire la trame réseau. Un effet direct des VLANs est une limitation des domaines de *brodacast* Ethernet. Nous allons examiner plusieurs scénarios basiques d'utilisation des VLANs en précisant les parts de configuration à la charge de la machine et à la charge du commutateur. Nous allons baser nos exemples sur des machines tournant sous **Debian** et un commutateur **Juniper** embarquant **JunOS** (un dérivé de **FreeBSD**).

Il faut bien comprendre qu'avant l'avènement des VLANs, un routeur **A** ne pouvait envoyer au commutateur **B** (non *manageable*) qu'un seul réseau et l'on devait, dès lors, avoir un deuxième lien physique vers un autre commutateur pour

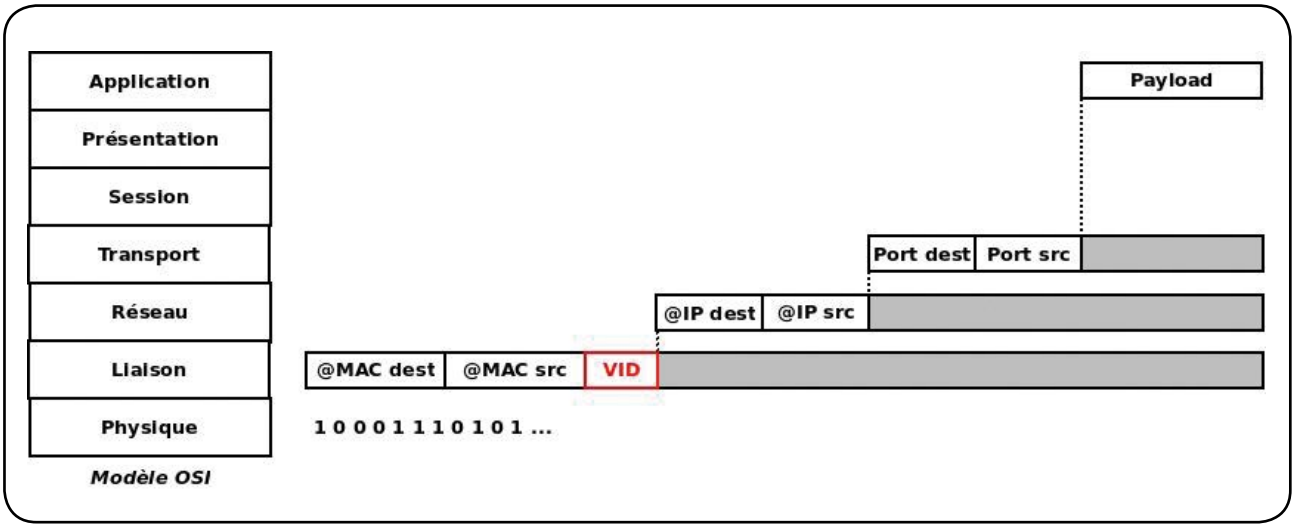


Fig. 2 : Intégration du VID dans la trame Ethernet

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

C'est simple... c'est possible sur :

http://www.ed-diamond.com

... OU SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE AU VERSO ET RENVOYEZ CE DOCUMENT COMPLET À L'ADRESSE CI-DESSOUS !

Ce document est la propriété exclusive de Johann Locatelli (johann.locatelli@businessdecision.com)

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

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

❑ Je souhaite recevoir les offres promotionnelles des partenaires des Éditions Diamond.

En envoyant ce bon de commande, je reconnais avoir pris connaissance des conditions générales de vente des Éditions Diamond à l'adresse internet suivante : http://**boutique.ed-diamond.com/content/3-conditions-generales-de-ventes** et reconnais que ces conditions de vente me sont opposables.

LINUX MAGAZINE / FRANCE

Les Éditions Diamond
Service des Abonnements
10, Place de la Cathédrale
68000 Colmar – France
Tél. : + 33 (0) 3 67 10 00 20
Fax : + 33 (0) 3 67 10 00 21

Vos remarques :

.....

.....

VOICI TOUTES LES OFFRES COUPLÉES AVEC GNU/LINUX MAGAZINE !

POUR LE PARTICULIER ET LE PROFESSIONNEL ...

Prix TTC en Euros / France Métropolitaine

CHOISISSEZ VOTRE OFFRE !

Offre	SUPPORT / ABONNEMENT (Prix en Euros / France Métropolitaine)	PAPIER Réf	PAPIER Tarif TTC	PAPIER + PDF (PDF 1 lecteur) Réf	PAPIER + PDF Tarif TTC	PAPIER + BASE DOCUMENTAIRE (1 connexion BD) Réf	PAPIER + BASE DOCUMENTAIRE Tarif TTC	PAPIER + PDF + BASE DOCUMENTAIRE (PDF 1 lecteur + 1 connexion BD) Réf	PAPIER + PDF + BASE DOCUMENTAIRE Tarif TTC
LM	11n° GLMF	LM1	65,-	LM12	95,-	LM13	149,-	LM123	174,-
LM+	11n° GLMF + 6n° HS	LM+1	118,-	LM+12	177,-	LM+13	197,-	LM+123	256,-
LES COUPLAGES « LINUX »									
A	11n° GLMF + 6n° LP	A1	95,-	A12	140,-	A13	218,-	A123	263,-
A+	11n° GLMF + 6n° HS + 6n° LP + 3n° HS	A+1	182,-	A+12	263,-	A+13	300,-	A+123	386,-
B	11n° GLMF + 6n° MISC	B1	100,-	B12	147,-	B13	233,-	B123	280,-
B+	11n° GLMF + 6n° HS + 6n° MISC + 2n° HS	B+1	172,-	B+12	248,-	B+13	300,-	B+123	381,-
C	11n° GLMF + 6n° LP + 6n° MISC	C1	135,-	C12	197,-	C13	312,-	C123	374,-
C+	11n° GLMF + 6n° HS + 6n° LP + 3n° HS + 6n° MISC + 2n° HS	C+1	236,-	C+12	339,-	C+13	403,-	C+123	516,-
LES COUPLAGES « EMBARQUÉ »									
F	11n° GLMF + 6n° HK* + 4n° OS	F1	125,-	F12	188,-	F13	229,-*	F123	292,-*
F+	11n° GLMF + 6n° HS + 6n° HK* + 4n° OS	F+1	183,-	F+12	275,-	F+13	287,-*	F+123	379,-*
LES COUPLAGES « GÉNÉRAUX »									
H	11n° GLMF + 6n° HK* + 6n° LP + 6n° MISC + 4n° OS	H1	200,-	H12	300,-	H13	402,-*	H123	499,-*
H+	11n° GLMF + 6n° HS + 6n° HK* + 6n° LP + 3n° HS + 6n° MISC + 2n° HS + 4n° OS	H+1	301,-	H+12	452,-	H+13	493,-*	H+123	639,-*

Les abréviations des offres sont les suivantes : LM = GNU/Linux Magazine France | HS = Hors-Série | LP = Linux Pratique | OS = Open Silicium | HC = Hackable

* HK : Attention : La base Documentaire de Hackable n'est pas incluse dans l'offre.

PRO OU PARTICULIER = CONNECTEZ-VOUS SUR : http://www.ed-diamond.com pour consulter toutes les offres !

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

envoyer un autre réseau. Autant dire que l'occupation d'un bâtiment au sens informatique du terme était avant toutes choses affaire de regroupement d'un maximum de personnes travaillant dans le même service, voire dans le pire des cas et faute de moyens la mise en commun de personnes sans aucune interaction les unes avec les autres mais contraintes de partager le même réseau. Assurer la sécurité du SI était alors plus que problématique.

Avec l'apparition du VLAN, le routeur envoie désormais les réseaux via un lien 802.1Q sur un même et unique média (fibre, cuivre, WiFi, etc.) vers les commutateurs. Ces derniers gérant les VLANs, peuvent dès lors avec leur matrice de commutations multi-vlan affecter à un port un vlan indépendamment de la configuration des autres ports. Le confinement ainsi assuré, des utilisateurs sans aucun lien peuvent désormais partager le même commutateur physique en ne se trouvant « que dans leur réseau ». Cette avancée technique a permis, outre une meilleure gestion des mètres carrés d'un bâtiment, de substantielles économies sur le tirage de rocades d'interconnexions et sur l'achat de commutateurs mutualisés.

2.1 Mode « access »

Le mode « access » propose de mettre un port du commutateur dans un VLAN donné. La figure 3 présente un exemple simple de découpage en VLAN configuré en port mode access. Dans cet exemple le commutateur dispose de 4 ports numérotés de **1** à **4**. Les ports **1** et **4** sont dans le VLAN identifiés par le VID **3** tandis que les ports **2** et **3** sont dans le VLAN possédant le VID **4**. La machine **A** est connectée au port **1**, les machines **B** et **C** sont connectées aux ports **2** et **3** et la machine **D** est connectée au port **4**. Le commutateur est donc divisé de façon logique en deux segments LAN. Ainsi la machine **A** ne peut parler qu'avec **D** et **B** ne peut parler qu'avec **C**. Voyons en pratique comment réaliser un telle configuration sur notre commutateur Juniper pour le port numéro **1** (c'est la même histoire pour les trois autres) :

```
ngreneche@sw-eth> configure
Entering configuration mode

{master:0}[edit]
ngreneche@sw-eth# set vlans VLAN003 vlan-id 3
ngreneche@sw-eth# set interfaces ge-1/0/1 unit 0 family ethernet-
switching port-mode access
ngreneche@sw-eth# set interfaces ge-1/0/1 unit 0 family ethernet-
switching vlan members VLAN003
```

Dans cette séquence de commandes, on commence par passer le commutateur en mode configuration. Ensuite on crée le VLAN. La chaîne de caractère **VLAN003** est libre : on peut mettre ce que l'on veut. En revanche, le paramètre **vlan-id** est l'identifiant numérique attaché au VLAN utilisé comme tag dans les trames. Nous allons maintenant configurer le port **1** du commutateur. Ce port est nommé **ge-1/0/1** dans la logique de numérotation Juniper. La première commande **set interface** passe le port du commutateur en mode access. La seconde définit le VLAN dans lequel ce port est positionné.

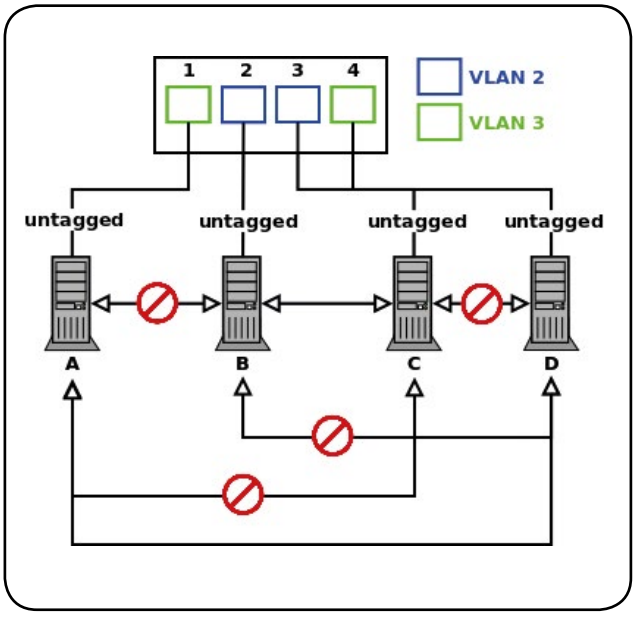


Fig. 3 : Commutateur mode « access »

2.2 Mode « 802.1Q »

Dans ce mode, le port du commutateur s'attend à recevoir des trames déjà taguées par l'émetteur. La configuration se fait donc à la fois côté commutateur et côté machine connectée. La terminologie pour un tel lien est multiple : « *port trunk* », « lien 802.1Q », etc. Nous allons voir comment configurer une interface de VLAN sur une machine Linux et comment passer un port en mode « *trunk* » sur notre commutateur Juniper. Le but de la manipulation est de multi-domicilier la machine **A** dans les VLANs **3** et **4**.

2.2.1 Configuration de la machine

Sous Linux, le support des VLANs dépend de deux parties : un module noyau nommé 8021q et une collection d'outils en espace utilisateur pour configurer les interfaces.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Le module noyau est disponible dans le système de base de Debian, il suffit de le charger :

```
root@A:~# modprobe 8021q
```

Pour rendre le chargement du module persistant au redémarrage il faut ajouter la ligne suivante dans le fichier **/etc/module** :

```
8021q
```

Pour la partie outils en espace utilisateurs, il faut installer le package **vlan** :

```
root@A:~# apt-get install vlan
```

Nous allons ensuite créer une interface pour le VLAN **3**. Cette interface réseau est une interface logique créée au-dessus d'une interface réelle (par exemple **eth0**). La seule chose à spécifier lors de la création de cette interface logique est l'identifiant de VLAN (le VID). Le but de cette interface est de taguer une trame avec pour identifiant le VID spécifié lorsqu'elle est émise et de retirer le tag (si le VID de la trame correspond au VID de l'interface de VLAN) pour le faire traiter par la pile réseau. Pour créer une interface de VLAN, il faut utiliser la commande **vconfig** :

```
root@A:~# vconfig add eth0 3
```

Cette commande va créer une interface **eth0.3**. Cette interface est une interface logique pour le VLAN **3** construite au-dessus de **eth0**. Pour l'activer :

```
root@A:~# ifup eth0.3
```

Il faut juste répéter ces étapes pour configurer une interface dans le VLAN **4**.

2.2.2 Configuration du commutateur

Côté commutateur, la donne a changé sur le port **1**. Il faut maintenant qu'il accepte les paquets tagués avec les VID **3** et **4**. Il faut donc reconfigurer le port **1** du commutateur en mode 802.1Q. Chez Juniper (comme chez Cisco d'ailleurs) ce mode est appelé « *trunk* » (voir figure 4) :

```
ngreneche@sw-eth# set interfaces ge-1/0/1 unit 0 family ethernet-
switching port-mode trunk
```

Cette étape n'est pas suffisante car nous voulons tout de même restreindre les VID pouvant transiter par le port. Il faut les spécifier en tant que *members* du lien 802.1Q sinon le port traitera tous les paquets tagués.

```
ngreneche@sw-eth# set interfaces ge-1/0/1 unit 0 family ethernet-
switching vlan members VLAN003
ngreneche@sw-eth# set interfaces ge-1/0/1 unit 0 family ethernet-
switching vlan members VLAN004
```

À présent, la machine **A** étant bi-domiciliée dans les deux VLANs et le port étant configuré pour accepter les paquets des deux VLANs, **A** peut communiquer avec **B** et **C**. Nous avons établi une liaison (au sens 2 du modèle OSI) entre les trois machines.

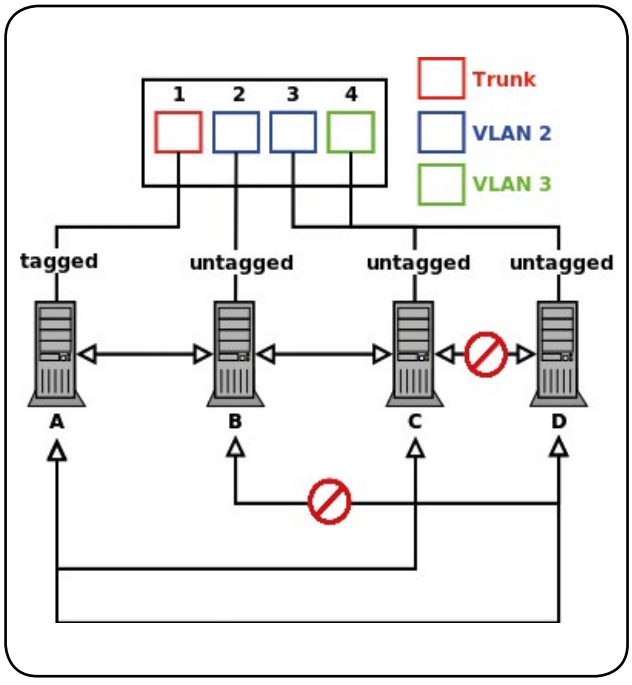


Fig. 4 : Commutateur mode « trunk »

2.3 Et le niveau réseau ?

Établir une liaison entre des machines n'a que peu d'intérêts si les processus tournant dessus ne peuvent pas communiquer. Or dans le monde actuel, les machines communiquent quasiment exclusivement en IP. Nous allons donc voir comment combiner le niveau liaison et réseau pour obtenir des réseaux IP cloisonnés logiquement. Une bonne pratique est de superposer un VLAN et un réseau IP. Dans notre exemple, nous pourrions décider que le VLAN **3** superpose le réseau IP **192.168.3.0/24** et que le VLAN **4**

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

superpose le **192.168.4.0/24**. Le but de la manipulation va être que les machines des deux VLANs puissent toutes communiquer entre elles en IP tout en appartenant à des VLANs différents. La machine **A** servira de passerelle entre les deux VLANs. Nous allons fixer la topologie réseau :

Machine	Interface	Adresse IP
A	eth0.3	192.168.3.254
	eth0.4	192.168.4.254
B	eth0	192.168.4.1
C	eth0	192.168.4.2
D	eth0	192.168.3.1

Pour les machines **B**, **C** et **D** il s'agit d'une configuration classique. Pour la machine **A** c'est un peu plus compliqué. Pour mémoire, **A** dispose de deux interfaces : **eth0.3** et **eth0.4**. L'interface **eth0.3** va prendre l'IP **192.168.3.254** et **eth0.4** l'IP **192.168.4.254** :

```
root@A:~# ifconfig eth0.3 192.168.3.254 netmask 255.255.255.0
root@A:~# ifconfig eth0.4 192.168.4.254 netmask 255.255.255.0
```

Examinons maintenant la table de routage de **A** :

```
root@A:~# netstat -anr
Table de routage IP du noyau
Destination  Passerelle  Genmask       Indic   MSS Fenêtre irtt Iface
0.0.0.0      X.X.X.X     0.0.0.0       UG        0 0          0 eth1
192.168.3.0  0.0.0.0     255.255.255.0 U         0 0          0 eth0.3
192.168.4.0  0.0.0.0     255.255.255.0 U         0 0          0 eth0.4
```

Les deux dernières règles sont des règles implicites qui ont été générées au moment où les interfaces **eth0.3** et **eth0.4** ont été configurées avec **ifconfig**. Ces deux règles signifient : pour toucher le réseau **192.168.3.0/24** passer par **eth0.3** et pour toucher le réseau **192.168.4.0/24** passer par **eth0.4**. Cela ne permet pas pour autant à un paquet de passer d'un réseau IP à l'autre. Il faut activer l'IP *forwarding* :

```
root@A:~# sysctl -w net.ipv4.ip_forward=1
```

Il faut également spécifier comme passerelle par défaut le **192.168.3.254** (**eth0.3**) pour les machines du VLAN **3** et le **192.168.4.254** (**eth0.4**) pour les machines du VLAN **4**.

À partir de ce moment, le transfert de paquets IPv4 est autorisé entre les interfaces de la machine. Quand une trame va arriver du VLAN **3** à destination d'une machine du VLAN **4**, le paquet va arriver sur la passerelle par défaut. Le domaine de *brodcast* est limité par les VLANs donc le *who-has* ARP échoue sur la machine du VLAN **3**, elle va donc envoyer la trame à sa passerelle par défaut : **eth0.3** de **A**. La pile réseau de **A** va examiner la destination du paquet. Cette destination est dans le VLAN **4**. L'IP *forwarding* étant activé, **A** va router le paquet vers l'interface **eth0.4** qui va faire un *who-has* ARP couronné de succès. **A** va donc encapsuler le paquet dans une trame à destination de la machine cible via **eth0.4**.

Pour inscrire cette configuration en dur, il faut ajouter les lignes suivantes dans le fichier **/etc/network/interfaces** de **A** :

```
auto eth0.3
iface eth0.3 inet static
  address 192.168.3.254
  netmask 255.255.255.0

auto eth0.4
iface eth0.4 inet static
  address 192.168.4.254
  netmask 255.255.255.0
```

Et aussi activer l'IP *forwarding* dans le fichier **/etc/sysctl.conf** :

```
net.ipv4.ip_forward=1
```

La configuration est maintenant persistante au redémarrage. Évidemment il est possible (et souhaitable) d'ajouter des règles de filtrage iptables sur **A** entre **eth0.3** et **eth0.4** pour éviter que n'importe quoi transite d'un VLAN à l'autre. On notera que le travail qui a été fait sur **A** est tout à fait réalisable directement sur le commutateur (selon ses capacités de management). Pour cela, il faut définir les interfaces de VLAN sur le commutateur :

```
ngreneche@sw-eth# set interfaces vlan unit 3 family inet address
192.168.3.254/24
ngreneche@sw-eth# set interfaces vlan unit 4 family inet address
192.168.4.254/24
```

Ensuite, selon ses capacités, le commutateur peut filtrer / router le trafic inter-VLANs (voir figure 5). Certains services peuvent aussi être déployés directement sur le commutateur attaché aux interfaces de VLAN (comme un serveur DHCP servant un *pool* dans la plage IP superposant le VLAN).

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

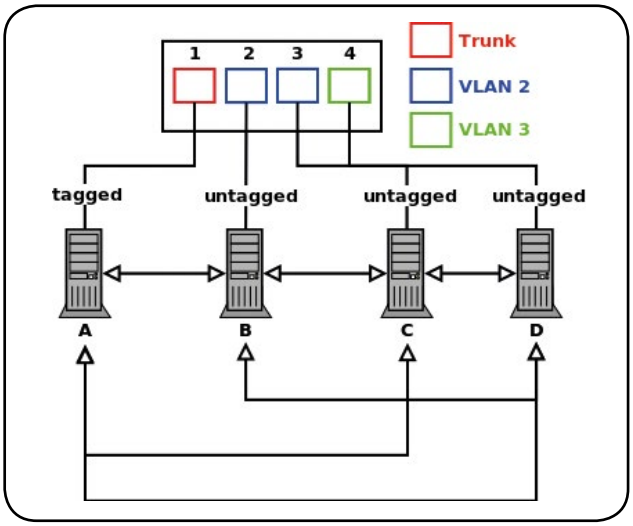


Fig. 5 : Routage inter-VLANs

3 Utilisation avancées des VLANs

Dans cette section nous allons voir une fonctionnalité méconnue des équipements de commutation : le « native VLAN ». Cette fonctionnalité est d'un grand secours lorsqu'une machine connectée au commutateur ne sait pas taguer une trame Ethernet et que l'interface du commutateur attend des paquets tagués (mode 802.1Q). Nous allons ensuite montrer comment les VLANs peuvent participer à l'interconnexion de réseau entre eux. Enfin, nous allons mentionner quelques outils construits au-dessus des VLANs.

3.1 Native VLAN

C'est un VLAN spécifique qui confine les trames non tagués. Autrement dit, sur un port 802.1Q les trames Ethernet non taguées par un ID sont placées dans le native VLAN. Ce VLAN va faire transiter l'ensemble des protocoles non tagués à l'instar de CDP, *Spanning Tree*, DTP, etc. Le VLAN *native* est également une option intéressante dans des cas spécifiques. Par défaut le VLAN native est le **1**. Il arrive cependant qu'il soit nécessaire de modifier son ID.

Prenons un réseau segmenté de façon logique en deux VLANs : un pour le trafic normal (**VLAN002**) des machines et un autre hors bande pour les contrôleurs IPMI (**VLAN003**). Les ports du commutateur doivent donc être configurés en mode 802.1Q ; ils attendent donc des paquets tagués. Supposons que les systèmes soient déployés via PXE. Les cartes réseau effectuant le démarrage en PXE sont pour la plupart incapables de taguer les trames. Grâce aux *natives* VLAN, il est possible de créer un VLAN de déploiement (VID 4) contenant les trames non taguées des systèmes en cours d'installation PXE. Voici la configuration des ports sur JunOS :

```
ngreneche@sw-eth# set interfaces interface-range ge-1/0/1 unit 0
family ethernet-switching port-mode trunk
ngreneche@sw-eth# set interfaces ge-1/0/1 unit 0 family ethernet-
switching vlan members VLAN002
ngreneche@sw-eth# set interfaces ge-1/0/1 unit 0 family ethernet-
switching vlan members VLAN003
ngreneche@sw-eth# set interfaces ge-1/0/1 unit 0 family ethernet-
switching native-vlan-id 4
```

3.2 VLAN d'interconnexion BGP

C'est un VLAN créé entre plusieurs équipements réseaux (c'est-à-dire que chacun des équipements réseau possède une interface dans ce VLAN). Prenons le cas du protocole de routage BGP, ce protocole fonctionne en définissant une chaîne de *next hop* appelés *neighbors*. Chacun de ces *neighbors* correspond en réalité à une interface d'un équipement de routage connecté au VLAN d'interconnexion. Un VLAN d'interconnexion n'est pas destiné à faire transiter du trafic réseau usuel (stream, web, mail, etc.) mais se présente plutôt comme un réseau hors bande. Dans ce cas l'utilisation des VLANs nous évite de déployer un réseau d'administration physique avec les problèmes de coût et de SPOF (*Single Point Of Failure*) inhérents.

Conclusion

Bien qu'assez anciens, les VLANs sont encore activement utilisés à la fois dans les environnements de production et expérimentaux. Ils permettent de segmenter des réseaux à moindres frais. Combinés aux capacités de châssis virtuels des équipements réseau, ils contribuent à calquer la topologie réseau de données sur la topologie organisationnelle de l'entité indépendamment de la répartition géographique des postes de travail. Les VLANs sont également utilisés comme briques par plusieurs services. Le plus connu, le 802.1X, concerne la sécurité réseau. Ce protocole permet d'authentifier les machines sur le réseau. Cependant, une solution 802.1X propose des services supplémentaires comme la vérification d'états des machines se raccordant au réseau (statut des mises à jour, dernière analyse antivirale, etc.). Si tous les paramètres (authentification et état) sont au vert alors la machine peut accéder au réseau, sinon elle est raccordée à un VLAN de remédiation où se trouvent uniquement les serveurs de mise à jour et antiviraux chargés de mettre la machine dans l'état attendu. ■

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Professionnels, Collectivités, R & D...

C'est possible ! Rendez-vous sur :

http://proboutique.ed-diamond.com

pour consulter les offres !

N'hésitez pas à nous contacter pour un devis personnalisé par e-mail : abopro@ed-diamond.com ou par téléphone : +33 (0)3 67 10 00 20

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

APPRENEZ À PROGRAMMER LA LIBPCAP

Arnaud FÉVRIER [Maître de conférences en Informatique, Aix Marseille Université, équipe eRISCS]

La star de l'analyse réseau, c'est wireshark ! Ses possibilités imposent l'admiration. Cependant, dans la vraie vie, l'administrateur a des questions auxquelles wireshark ne répondra pas ou difficilement. Est-ce que des mots de passe ou fichiers transitent en clair ? Les sessions TCP sont-elles toutes autorisées ? Il existe d'autres logiciels qui permettent de répondre simplement à ces questions. Nous souhaitons aller encore plus loin dans l'étude de nos réseaux : programmons les sondes réseaux !

Mots-clés : libpcap, tcpdump, wireshark, forensics, sécurité, analyse réseau, programmation réseau

Résumé

Nous présentons comment coder une application qui utilise la libpcap. Nous présentons les quelques fonctions de la bibliothèque qui permettent de réaliser un programme simple, puis nous montrons un exemple de programme et enfin, nous présentons quelques éléments pour étudier un paquet.

Un bon administrateur réseau va utiliser de nombreuses applications qui lui permettent de surveiller et corriger son réseau. Parmi les premières applications, il va utiliser des *sniffers* (**tshark**, **tcpdump**). Ensuite, il utilisera des outils pour mesurer la charge du réseau (**iptraf**). Puis, il pourra utiliser des détecteurs d'intrusion (**snort**), des détecteurs de vulnérabilités, des logiciels d'attaques (**yersinia**) ou de *crackage* de mots de passe (**john the ripper**) et bien d'autres.

Après avoir utilisé la panoplie classique, les logiciels libres fournissent aussi des programmes moins courants qui permettent, par exemple, de capturer les images transitant par le réseau (**driftnet**) ou les sessions TCP (**chaosreader**). Mais les logiciels existants ne peuvent répondre à toutes les questions. Dans ce cas, il peut être utile de développer son propre programme. La librairie libpcap, la base de tcpdump et des autres logiciels de capture facilitent grandement l'écriture d'une application spécifique.

Nous allons donc montrer comment développer une application de capture réseau. Nous présentons la structure d'une application utilisant la libpcap, puis nous présentons un exemple de programme. Enfin, nous donnons quelques idées pour analyser un paquet réseau.

1 Le squelette d'une application

Les applications utilisant la libpcap doivent toutes suivre la même séquence :

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

1. Il faut décider l'interface de capture. La librairie peut fournir une interface par défaut.
2. Ensuite, il faut initialiser la libpcap pour définir sur quelle(s) interface(s) écouter.
3. Éventuellement, mettre en place un filtre de capture de trafic réseau.
4. Enfin, nous entrons dans la boucle principale d'attente d'événements. À chaque paquet reçu, la boucle appelle une fonction *callback* qui va traiter le paquet (ou non).
5. Pour finir, il faut terminer la session.

1.1 Interface d'écoute

La première étape consiste à déterminer sur quelle interface la capture va s'effectuer. Dans le cas d'une interface Ethernet ou wifi, son nom ressemblera à **eth0**, **wlan0**, **wifi0**. Ces noms devraient bientôt changer avec le nouveau noyau. La commande **tshark -D** affiche la liste des interfaces sur lesquelles la capture peut être effectuée. Il est possible d'écouter sur les interfaces Ethernet, bien sûr, mais aussi sur de nombreuses autres interfaces réseaux, comme X25, RNIS, GPRS, Bluetooth ; et également sur d'autres supports, plus ou moins éloignés des réseaux comme l'USB, l'infrarouge, la TNT, les cartes à puces sans contact, le bus système (D-Bus) ou le bus automobile CAN. La librairie propose une fonction qui fournit un nom d'interface par défaut. Il s'agit vraisemblablement de l'interface utilisée par la station (Ethernet ou wifi). Après avoir choisi l'interface d'écoute, nous allons l'initialiser.

1.2 Initialisation

C'est la fonction **pcap_open_live** qui va effectuer l'initialisation de l'interface de capture. Cette fonction va éventuellement passer l'interface en mode immoral (*promiscuous*). Elle retourne une poignée (*handle*) : c'est une valeur identifiant l'interface ouverte. Nous allons maintenant préparer un filtre pour limiter les paquets capturés.

1.3 Filtre

Comme pour **tcpdump** ou **tshark**, il est possible d'écrire un filtre pour limiter les paquets capturés. La syntaxe est celle de **tcpdump**. Le filtre va créer une machine à états finis qui va grandement optimiser le temps de filtrage. Une fois le filtre généré, il faut l'appliquer à l'interface initialisée. Nous pouvons maintenant passer dans la boucle principale.

1.4 Boucle

Beaucoup de programmes, souvent de type serveur, entrent dans une boucle d'attente d'événements. Lors de l'arrivée d'un de ces événements une fonction est appelée, c'est la *callback*. La boucle principale va donc attendre l'arrivée d'un paquet. Si celui-ci traverse le filtre, alors la *callback* va être appelée avec en paramètres le paquet binaire et quelques informations sur le paquet (longueur, horodatage).

1.5 Terminaison

La boucle peut se terminer de plusieurs façons :

- Le nombre de paquets à capturer. La boucle terminera après les avoir reçus et le programme va continuer.
- La *callback* peut demander la fin de la boucle. Le contrôle est alors rendu à l'instruction qui suit la boucle.
- Un mécanisme classique consiste à interrompre le programme par la touche **<Ctrl>** + **<c>**. Pour cela, il faut ajouter une fonction qui va être appelée lors de la réception du signal. Ceci peut être utilisé pour constituer des statistiques ou un rapport sur l'ensemble de la capture. Le programme peut alors terminer.

2 Notre premier programme

Pour compiler une application, il faut avoir installé les outils essentiels de construction de programmes (compilateur, analyseurs syntaxiques, débuggeurs, etc.). Avec Debian, il faut donc installer **build-essential**. Il faut aussi installer le paquet de développement de la librairie : **apt-get install libpcap0.8-dev**. Le programme doit utiliser (au moins) le fichier d'en-tête **pcap.h** et ceux qui seront utilisés dans le corps du programme, comme **stdio.h**, par exemple :

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

Le C n'étant pas un langage très rigoureux, nous devons utiliser un débuggeur. Le couple **Emacs** + **gdb** fournit un environnement de choix, mais il existe d'autres environnements de développement. Un utilisateur normal n'a pas le droit de manipuler directement les interfaces réseaux. Une façon triviale (et périlleuse) consiste à tout faire dans l'environnement root, mais il existe des façons plus fines d'autoriser uniquement la capture réseau à un programme. Une solution consiste à fournir à l'exécutable les capacités utiles :

```
# setcap cap_net_raw,cap_net_admin=eip capture
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Les capacités étant perdues à chaque recompilation, j'utilise une boucle shell (**watch** ou **while**) pour remettre en permanence les capacités. Bien entendu, ceci est effectué par l'administrateur. Nous utiliserons **gcc**, GNU **make** et **gdb**. Voici le **Makefile** avec les options de compilation utilisées :

```
CFLAGS=-g -Wall -Werror

all: capture opendev

capture: capture.o
gcc -g -o $@ $< -lpcap
```

Pour débugguer un programme qui utilise des capacités spéciales, **gdb** doit être lancé par l'administrateur. Pour qu'il soit lancé par l'IDE, il faut soit faire un **sudo gdb**, soit un *setuid root* sur l'exécutable. Je recommande donc de faire le debug uniquement sur une machine (virtuelle ?) dédiée à cet usage. Nous allons maintenant choisir l'interface de capture.

2.1 Choix de l'interface

Le choix de l'interface est assez simple. Il s'agit d'obtenir une chaîne de caractères qui représente l'interface. Il s'agit du nom fourni par la commande **ip addr**. Le programme le plus élémentaire est le suivant :

```
#include <stdio.h>
#include <pcap.h>
/* ================================================== */
char * interfacepardefaut ()
{
  char * dev;
  char errbuf[PCAP_ERRBUF_SIZE];

  dev = pcap_lookupdev(errbuf);
  if (dev == NULL) {
    fprintf(stderr, "Couldn't find default device: %s\n", errbuf);
  }
  return(dev);
}
/* ================================================== */
int main(int argc, char *argv[])
{
  char * dev;

  if ( argc == 1)
    {
      dev = interfacepardefaut ();
    }
  else
    {
      dev = argv[1];
    }
  printf("Interface: %s\n", dev);
  return(0);
}
```

La boucle **main** teste le nombre d'arguments : s'il n'y a pas d'arguments, alors c'est l'interface par défaut, élue par la libpcap. Sinon, c'est le premier argument. Nous n'améliorerons pas ici l'analyse des arguments. La fonction **interfacepardefaut**, sans argument, renvoie une chaîne de caractères. La libpcap fournit le nom de l'interface par défaut. Nous voyons ici comment la libpcap traite les erreurs. Quand une fonction de la librairie détecte une erreur, elle écrit dans un *buffer* une chaîne de caractères expliquant l'erreur. Le programme affiche donc la chaîne de caractères entrée en paramètre ou le nom du périphérique par défaut. Nous allons maintenant (essayer d') initialiser l'interface.

2.2 Initialisation

Pour initialiser l'interface, nous allons utiliser la fonction **pcap_open_live**. Elle renvoie un pointeur permettant d'utiliser l'interface et prend comme paramètres cinq arguments :

- **dev** : le nom de l'interface ;
- **snapshot-length** : le nombre d'octets capturés pour chaque paquet. Le contenu de certains paquets peut donc être tronqué. Libpcap propose par défaut la taille **BUFSIZ** (**8192** octets aujourd'hui) ;
- **promisc** : indique si l'interface doit passer en mode *promiscuous*. Si l'interface n'est pas dans ce mode, alors seuls les paquets concernant l'interface seront capturés ;
- **Timeout** : spécifie une limite sur le temps de lecture ;
- **errbuf** : une chaîne de caractères indiquant le problème rencontré par la libpcap.

La fonction **main** du programme devient (les **...** indiquent que le code précédent n'est pas modifié) :

```
...
pcap_t *handle;
char errbuf[PCAP_ERRBUF_SIZE];
...
handle = pcap_open_live(dev, BUFSIZ, 1, 1000, errbuf);
if (handle == NULL) {
  fprintf(stderr, "Impossible d'initialiser %s parce que %s\n",
dev, errbuf);
  return(-1);
 }
...
```

Si le programme a la capacité réseau, il ne semble pas échouer. Pour vérifier, il est possible d'utiliser la commande magique **tail -f /var/log/syslog** pour vérifier que l'initialisation s'est bien passée :

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

ACTUELLEMENT DISPONIBLE

HACKABLE N°15 !

CONTRÔLEZ VOTRE ARDUINO DEPUIS VOTRE SMARTPHONE !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
... kernel: [350081.268390] device eth0 entered promiscuous mode
... kernel: [350081.280855] device eth0 left promiscuous mode
```

Nous allons maintenant préparer la boucle d'attente avant de mettre en place le filtre de capture.

2.3 Boucle d'attente d'événements

L'ajout de la boucle est assez facile. Nous allons utiliser la fonction **pcap_loop** qui retourne zéro si la fin prévue est atteinte et une valeur négative sinon. Ses paramètres sont :

- **handle** : la référence sur l'interface initialisée ;
- **cnt** : le nombre de paquets à traiter, **0** ou **-1** indiquant une capture infinie ;
- **fonction** : il s'agit d'un pointeur sur la fonction qui va traiter chaque paquet ;
- **NULL** : une donnée utilisateur qui peut être transmise à la fonction appelée. Nous n'en aurons pas besoin ici.

La fonction *callback* a trois paramètres en arguments :

- **user** : la donnée utilisateur transmise par la boucle, ici **NULL** ;
- **pkthdr** : un pointeur sur une structure donnant l'estampille temporelle et la longueur du paquet ;
- **paquet** : un pointeur sur le paquet.

Les structures ne doivent pas être libérées dans la fonction de *callback* et si les informations doivent être utilisées plus tard, alors il faut les dupliquer. La fonction que nous définissons ne traitera pas les paquets, mais affichera un simple **"bip"** à chaque paquet capturé.

```
void traitepaquet(u_char *args,
                  const struct pcap_pkthdr *header,
                  const u_char *packet)
{
  printf ("bip \n");
}
```

Nous modifions la fin de la fonction **main** pour utiliser la boucle :

```
res = pcap_loop(handle, 10 ,traitepaquet,NULL);
return(res);
```

Nous voyons un **bip** pour chaque paquet qui passe par l'interface. Selon l'utilisation, cela peut être plus ou moins rapide, au bout de dix paquets le programme termine.Nous allons maintenant ajouter un filtre de capture.

2.4 Filtre de capture

Nous supposons que le lecteur est un adepte de **tshark**, **tcpdump** ou autre et qu'il a l'habitude des filtres de capture de paquets. Sinon, le manuel de **tcpdump** fournira les renseignements utiles. La librairie libpcap va utiliser une chaîne de caractères, similaire aux arguments de **tcpdump** pour compiler un filtre avec la fonction **pcap_compile**. Puis, on applique le filtre à l'interface choisie en utilisant la fonction **pcap_setfilter**.

Les arguments sont les suivants :

- **handle** : le descripteur de l'interface ;
- **fp** : un pointeur sur une structure décrivant le filtre ;
- **filter_exp** : la chaîne de caractères spécifiant le filtre. Nous utiliserons le filtre port **4321** pour l'exemple ;
- **optimize** : spécifie si le code généré doit être optimisé (pourquoi ne le serait-il pas ?) ;
- **masque** : le masque réseau. Ceci n'est pas utile ici. Nous utiliserons **PCAP_NETMASK_UNKNOWN**.

Nous ajoutons la déclaration des variables au début du **main** et ces deux fonctions juste avant la boucle :

```
struct bpf_program fp;
char filter_exp[] = "port 4321";
bpf_u_int32 mask= PCAP_NETMASK_UNKNOWN ;
...
if (pcap_compile(handle, &fp, filter_exp, 0, mask) == -1) {
  fprintf(stderr, "Couldn't parse filter %s: %s\n",
          filter_exp, pcap_geterr(handle));
  return(-1);
}
if (pcap_setfilter(handle, &fp) == -1) {
  fprintf(stderr, "Couldn't install filter %s: %s\n",
          filter_exp, pcap_geterr(handle));
  return(-1);
}
...
```

Le traitement d'erreur se fait si une des fonctions renvoie **-1** et utilise la fonction **pcap_geterr**.

2.5 Clôture de la session

La boucle pcap peut terminer sur plusieurs conditions. Indiquer le nombre de paquets, c'est facile, mais en général, la capture est lancée sans savoir le critère d'arrêt. Les logiciels de capture terminent donc sur une interruption entrée

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

au clavier par la pression de la touche **<Ctrl> + <c>**. Ceci fonctionne avec notre code, mais le traitement s'arrête alors. Pour pouvoir faire un traitement final, nous allons utiliser les signaux Linux. Ces signaux permettent d'envoyer des interruptions depuis un processus vers un autre. Les signaux les plus connus sont ceux envoyés depuis un interpréteur de commandes vers un processus qu'il a lancé (**<Ctrl> + <c>**, **<Ctrl> + <z>**) ou par la commande mal nommée **kill** qui ne tue pas forcément ses victimes. Pour cela, nous allons devoir écrire une fonction déclenchée lors de la réception d'un signal, indiquer au système que nous souhaitons traiter les interruptions, ajouter les fichiers d'en-tête utiles et enfin afficher le numéro du processus utilisé. Nous avons besoin des en-têtes suivants :

```
#include <signal.h>
#include <unistd.h>
```

La fonction de traitement des interruptions sera minimale. Nous affichons juste une ligne :

```
void sig_handler(int signo)
{
  if (signo == SIGINT)
    printf("\nRéception de SIGINT\n");
}
```

Puis, en début de **main**, nous affichons le numéro de **pid** et déclarons le traitement local des interruptions :

```
pid_t PID;

PID=getpid();
printf("Pid = %d\n",PID);
if (signal(SIGINT, sig_handler) == SIG_ERR)
  fprintf(stderr,"\nErreur: impossible de récupérer SIGINT\n");
```

Nous avons maintenant les briques de base qui permettent de faire des applications complexes. L'utilisation des signaux peut permettre, par exemple, d'afficher des statistiques temporaires (comme le **SIGUSR1** pour **dd**) ou de faire un traitement final. Nous allons maintenant commencer à regarder l'intérieur des paquets.

3 Manipulation des paquets

Nous avons vu comment mettre en place la structure d'une application de la libpcap et quelques éléments annexes. Compter les paquets, c'est bien, mais ce n'est pas suffisant. Nous allons étudier leur structure.

3.1 Capture Ethernet

La libpcap sait capturer de nombreux réseaux. Le plus connu est Ethernet ; il y a près de 90 types dont la liste est décrite dans **[2]**. Pour vérifier si le réseau est bien un réseau Ethernet, il suffit d'inclure le code suivant :

```
if (pcap_datalink(handle) != DLT_EN10MB) {
    fprintf(stderr, "L'interface %s n'est pas  Ethernet\n",
dev);}
```

Le **10** de **10MB** est maladroit : tous les réseaux Ethernet vont répondre la même valeur.

3.2 Les métadonnées du paquet

La librairie fournit deux variables à notre fonction *callback*. La première variable s'appelle **header** et contient l'estampille temporelle du paquet, la longueur des données capturées et la longueur du paquet. Pour voir la différence entre ces deux valeurs, nous allons modifier l'appel à la fonction **pcap_open_live** pour réduire les données capturées à **12** octets.

```
handle = pcap_open_live(dev, 12, 1, 1000, errbuf);
```

Ensuite, nous utilisons deux variables statiques : le numéro du paquet et la date du premier paquet. Ainsi, nous pourrons afficher des statistiques sur chaque paquet. Le code de la fonction *callback* devient :

```
void traitepaquet(u_char *args, const struct pcap_pkthdr *header,
    const u_char *packet)
{
  static int packet_cnt = 1;
  static int packet_orig;
  struct timeval ts;
  bpf_u_int32 caplen;
  bpf_u_int32 len;
  ts= header->ts;
  caplen = header->caplen;
  len = header-> len;

  if (packet_cnt == 1){packet_orig = ts.tv_sec;}
  printf ("paquet %d (%d s) %d/  %d\n",
    packet_cnt, (int)ts.tv_sec - packet_orig, caplen, len);
  packet_cnt++;
}
```

La structure **timeval** est composée de deux entiers. L'un contient la date (à la seconde) et l'autre le nombre de microsecondes. Les dates sont des dates Unix. Pour afficher

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

la date dans un format humain, il est possible d'utiliser **date --date=@1461516474**. Nous allons afficher les secondes depuis le premier paquet. Nous obtenons des données statistiques :

```
Pid = 2343
Interface: eth0
paquet 1 (0 s) 12/  74
paquet 2 (0 s) 12/  74
...
paquet 16 (21 s) 12/ 66
paquet 17 (21 s) 12/ 777
```

Nous voyons que nous n'avons que les 12 premiers octets de chaque paquet. La différence entre **caplen** et **len** indique si un paquet a été tronqué. Repassons **pcap_open_live** à la valeur **BUFSIZ** pour pouvoir aller plus loin.

3.3 Analyse des paquets Ethernet

Pour analyser les paquets, il va falloir connaître plus en détail la structure d'un paquet réseau et vérifier si le type est bien le type attendu. Si le paquet est un paquet Ethernet, alors il va avoir des caractéristiques, comme l'adresse source ou destination. Nous capturons des paquets dont nous allons examiner le contenu. Pour manipuler les structures réseaux, nous allons ajouter quelques fichiers d'entête supplémentaires :

```
#include <netinet/in.h>
#include <netinet/if_ether.h>
#include <stdlib.h>
```

Les adresses internet sont constituées de 6 octets : il faut une fonction qui affiche ces 6 octets. Nous définissons donc une fonction **mac2str** qui convertit un tableau de 6 octets en une chaîne de caractères :

```
char * mac2s(u_int8_t * etheraddr)
{
  char*  res;

  res =malloc(ETH_ALEN+2);
  sprintf(res, "%x:%x:%x:%x:%x:%x",
  (unsigned int)etheraddr[0], (unsigned int)etheraddr[1],
  (unsigned int)etheraddr[2], (unsigned int)etheraddr[3],
  (unsigned int)etheraddr[4], (unsigned int)etheraddr[5] );
  return res;
}
```

Bon, le **malloc** sans le **free** nous facilite le travail au prix d'une dérive mémoire. Comme le programme est un logiciel pédagogique, ça ne devrait pas poser de problèmes. Il ne reste plus qu'à modifier la fonction *callback* pour récupérer les informations. **net/ethernet.h** fournit la structure de données pour l'en-tête Ethernet. Le paquet commençant par celui-ci, un *cast* permet d'utiliser la structure :

```
struct ether_header *eptr;
...
eptr = (struct ether_header *) packet;
printf("Ethernet type= %d(%d) IP:%d\n",
       ntohs (eptr->ether_type),
       eptr->ether_type,ETHERTYPE_IP);
printf(" Source: %s -> Destination: %s \n",
       mac2s(eptr->ether_shost), mac2s(eptr->ether_dhost));
```

Ce code affiche le type ethernet (IP, ARP, etc.) en utilisant la fonction **ntohs** (*network to home system*) pour changer éventuellement l'ordre des octets dans un mot mémoire et la valeur attendue pour un type IP. D'autres valeurs sont possibles comme **0x86dd** pour IPv6 ou **0x0806** pour ARP. Il est aisé de vérifier par **ip addr** que l'adresse *mac* de la carte réseau est bien affichée.

Conclusion

Pour répondre à des questions spécifiques, il est parfois plus facile d'attaquer directement la programmation de la libpcap que d'utiliser la galaxie tcpdump/tshark/wireshark. Nous avons donc un outil de plus à notre disposition. Il ne faut cependant pas oublier les possibilités offertes de scripter wireshark ou tshark pour arriver à des résultats similaires. Le très grand nombre d'applications libres utilisant la libpcap fournit des exemples de programmes qui peuvent grandement accélérer le développement.

Il existe de nombreuses librairies sous GNU/Linux qui offrent des fonctionnalités de manipulation avancées, comme la **libnet** par exemple.

Enfin, je n'ai pas trouvé la meilleure méthode pour débugger en tant qu'utilisateur normal un programme qui doit avoir des privilèges avancés. ■

Références

[1] Le site de la libpcap : http://www.tcpdump.org/

[2] La liste des types de réseaux que la libpcap peut gérer : http://www.tcpdump.org/linktypes.html

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

ACTUELLEMENT DISPONIBLE

MISC N°88 !

WEB : QUELLES ÉVOLUTIONS POUR LA SÉCURITÉ ?

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

FORMAT MIDI ET MUSIQUE ALGORITHMIQUE

Vincent MAGNIN [Maître de conférences à Polytech Lille et l'IEMN, administrateur du projet gtk-fortran]

Avec le format MIDI, vous pouvez mêler les plaisirs de la composition musicale et de la programmation. Dans ce second article, nous allons explorer quelques aspects de la composition algorithmique.

Mots-clés : Format MIDI, musique algorithmique, musique stochastique, langage C

Résumé

Dans ce second article consacré au format MIDI, nous utilisons notre programme en C pour composer un canon à trois voix avec une basse continue, à partir du début du canon de Pachelbel puis nous explorons la musique stochastique en parcourant aléatoirement une gamme de blues. C'est également l'occasion de découvrir l'usage des percussions dans le format MIDI.

Lors du précédent article **[1]**, nous avions écrit un programme en C permettant de créer des fichiers au format MIDI (*Musical Instrument Digital Interface*). Nous allons maintenant l'utiliser pour explorer d'une part les possibilités offertes par MIDI et d'autre part certains liens pouvant exister entre musique et algorithmique. Une partition musicale comporte en effet des instructions (notes, nuances, etc.), des répétitions et des branchements conditionnels. Il s'agit donc d'un algorithme.

Les fichiers sources sont disponibles sur le dépôt GitHub du magazine. Rappelons que le fichier `midi.c` contient toutes les procédures et fonctions qui seront utilisées par les différents exemples développés dans cet article. Pour les fichiers `.mid` et `.mp3` des exemples sont disponibles sur mon site **[2]**, ainsi que quelques bonus.

1 Canon de Pachelbel

1.1 Introduction

Afin de vous donner envie de vous lancer dans la programmation MIDI et d'aller jusqu'au bout de cet article, il me faut de l'artillerie lourde. Le canon de Pachelbel devrait faire l'affaire **[3]**. Nous allons donc nous faire la main avec les premières mesures de ce canon en ré majeur composé vers 1700. Il est composé d'une basse continue et obstinée qui répète huit noires. Au bout de deux mesures, un premier violon commence à jouer un thème composé de seize noires. Un deuxième violon commence ce même thème deux mesures plus loin, puis un troisième violon encore deux mesures plus loin. C'est ce que l'on appelle un canon à trois voix (voir figure 1), procédé simple, à la structure algorithmique limpide, mais du plus bel effet !

Je vous rappelle que la première piste d'un fichier MIDI contient normalement les métadonnées. Et dans la procédure `ecrire_piste1()`, nous allons fixer un tempo de 60 (noires par minute), c'est-à-dire qu'une noire durera un million de micro-secondes (ligne 7) :

```
03: #include "midi.c"
04:
05: void ecrire_piste1(FILE *fichier) {
06:     unsigned long marque = MIDI_ecrire_en_tete_piste(fichier) ;
07:     MIDI_tempo(fichier, 1000000) ;
08:     MIDI_fin_de_la_piste(fichier) ;
09:     ecrire_taille_finale_piste(fichier, marque) ;
10: }
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

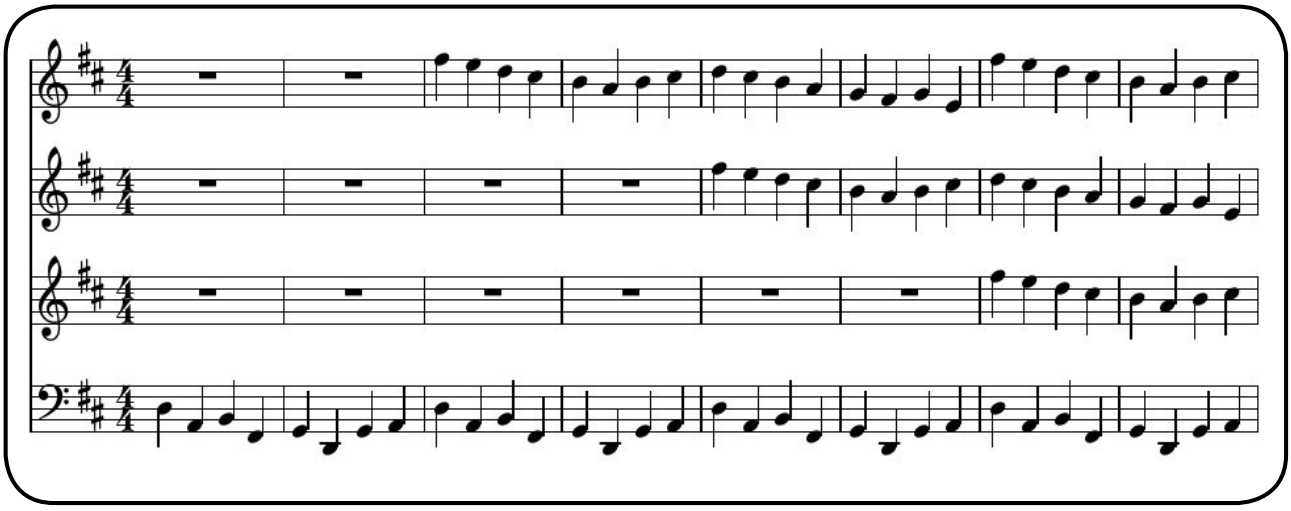


Fig. 1. Partition de notre fichier MIDI obtenue avec le logiciel Rosegarden.

1.2 Basse continue

Définissons tout d'abord la piste de la basse obstinée. Ligne 15, nous mettons un effet de réverbération moyen puisque le niveau est de **64** alors que le maximum possible est de **127**. Ligne 16, nous choisissons l'instrument General Midi **48** (*String Ensemble 1*) parmi les 128 disponibles **[4]**. Rappelons que la numérotation informatique des instruments commence à zéro. Nous définissons ensuite un tableau d'octets **basse[]** qui contient les numéros MIDI des huit notes à répéter **[5]** (la première, **50**, est un *ré2*). La boucle sur **i** écrit les huit noires dans le fichier, avec un volume sonore moyen **64**. La boucle sur **j** répète trente fois ces huit notes (ces deux mesures). Je vous l'ai dit qu'elle était obstinée, cette basse !

```
13: void ecrire_piste_basse(FILE *fichier) {
14:   unsigned long marque = MIDI_ecrire_en_tete_piste(fichier) ;
15:   MIDI_Control_Change(fichier, 0, reverb, 64) ;
16:   MIDI_Program_Change(fichier, 0, 48) ;
17:
18:   unsigned char basse[8] = {50, 45, 47, 42, 43, 38, 43, 45} ;
19:   for(int j = 1 ; j <= 30 ; j = j + 1) {
20:      for(int i = 0 ; i < 8 ; i = i + 1){
21:         Note_unique_avec_duree(fichier, 0, basse[i], 64, noire) ;
22:      }
23:   }
24:   MIDI_fin_de_la_piste(fichier) ;
25:   ecrire_taille_finale_piste(fichier, marque) ;
26: }
```

1.3 Canon à trois voix

La procédure **ecrire_pistes_canon()** ci-dessous va créer les trois pistes (voix) du canon, numérotées ici 3, 4 et 5. La ligne 36 permet de décaler le démarrage de chaque voix de huit noires par rapport à la précédente, en écrivant une note avec un volume sonore nul (il s'agit donc d'une pause). Les seize notes du thème sont stockées dans le tableau **theme[]**. La première note, **78**, est un *fa#4*. La boucle sur **i**, la plus interne, écrit le thème. La boucle sur **j** va écrire ce thème quinze fois sur chaque piste mais, petit raffinement, pour donner plus d'ampleur à notre morceau sans trop d'effort, la ligne 38 va permettre de changer d'instrument à chaque fois, en puisant dans une liste d'instruments MIDI bien choisis stockés dans le tableau **instrument[]**. On commencera donc par les instruments 40, 41 et 42 (violon, alto et violoncelle), puis on glissera progressivement vers les autres instruments. La boucle sur **piste** s'occupe bien sûr de créer les trois voix.

```
29: void ecrire_pistes_canon(FILE *fichier) {
30:     unsigned char instrument[17] = {40, 41, 42, 44, 45, 48,
49, 51, 52, 89, 90, 91, 92, 94, 95, 99, 100} ;
31:     unsigned char theme[16] = {78, 76, 74, 73, 71, 69, 71,
73, 74, 73, 71, 69, 67, 66, 67, 64} ;
32:
33:     for(int piste = 3 ; piste <= 5 ; piste = piste + 1) {
34:         unsigned long marque = MIDI_ecrire_en_tete_
piste(fichier) ;
35:         MIDI_Control_Change(fichier, piste, reverb, 64) ;
36:         Note_unique_avec_duree(fichier, piste, 0, 0,
8*noire*(piste - 2)) ;
37:         for(int j = 0 ; j < 15 ; j = j + 1) {
38:             MIDI_Program_Change(fichier, piste,
instrument[(piste - 3) + j]) ;
39:             for(int i = 0 ; i < 16 ; i = i + 1){
40:                 Note_unique_avec_duree(fichier, piste, theme
[i], 64, noire) ;
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
41:             }
42:         }
43:         MIDI_fin_de_la_piste(fichier) ;
44:         ecrire_taille_finale_piste(fichier, marque) ;
45:     }
46: }
```

Dans le programme principal, nous indiquons ligne 50 que le fichier MIDI comportera cinq pistes (une pour les métadonnées, une pour la basse obstinée et trois pour le canon) :

```
48: int main(void) {
49:     FILE *fichier_midi = fopen("canon.mid", "wb") ;
50:     MIDI_ecrire_en_tete(fichier_midi, 1, 5, noire) ;
51:     ecrire_piste1(fichier_midi) ;
52:     ecrire_piste_basse(fichier_midi) ;
53:     ecrire_pistes_canon(fichier_midi) ;
54:     fclose(fichier_midi) ;
55: }
```

1.4 Music Non Stop !

Après compilation et exécution, vous pouvez utiliser le logiciel **TiMidity++** (paquet **timidity** dans le cas d'une distribution Ubuntu) **[6]** pour le jouer :

```
$ gcc canon. c
$ ./a.out
$ timidity canon.mid
```

Mais pour un meilleur rendu, je vous conseille fortement d'utiliser la fonte sonore *Fluid R3 General MIDI soundfont* disponible dans le paquet **fluid-soundfont-gm** (142 Mio) sous Ubuntu, d'autant plus que certains instruments n'existent pas dans la fonte par défaut :

```
$ timidity canon.mid -x "soundfont /usr/share/sounds/sf2/FluidR3_
GM.sf2"
```

Si vous avez un instrument avec une entrée MIDI, vous pouvez bien sûr envoyer le flux dessus avec la commande ci-dessous (en supposant que votre instrument soit sur le port MIDI **20**) :

```
$ aplaymidi -p 20 canon.mid
```

Je vous renvoie pour plus de détails sur ce point au hors-série n°29 « Musique & Son » de Linux Pratique **[7]**.

Pas mal pour un fichier MIDI de 8902 octets ! Attention, ces 4'30" de musique peuvent devenir obsédantes... Cette durée me fait penser que pour vous reposer le cerveau et les oreilles, vous pourrez ensuite réfléchir à un algorithme pour jouer les trois mouvements du morceau *« 4'33" »* de John Cage :-).

1.5 Variations

Dans les trois voix de notre canon, les deuxième et troisième sont des *imitations strictes* de la première. Mais une imitation peut également être par exemple rétrogradée (dite *à l'écrevisse*), c'est-à-dire que le thème est joué de la fin vers le début, *en miroir* (on inverse le mouvement vertical de la mélodie), *transposée* (par exemple à la quinte), etc. Dans la pratique, l'effet n'est pas toujours heureux (apparition de dissonances plus ou moins marquées) et nécessite de la recherche. Je vous propose de remplacer la ligne 40 de notre programme par la structure de choix suivante :

```
01: switch(piste) {
02:   case 3:
03:     Note_unique_avec_duree(fichier, piste, theme[i], 64,
noire) ;
04:     break ;
05:   case 4:
06:     Note_unique_avec_duree(fichier, piste, theme[15-i], 64,
noire) ;
07:     break ;
08:   case 5:
09:       if (i%2 == 0) Note_unique_avec_duree(fichier, piste,
theme[i], 64, 2*noire) ;
10: }
```

La deuxième voix est une imitation à l'écrevisse (**15-i**). La troisième ne joue qu'une note sur deux mais pendant la durée d'une blanche (deux noires). Le résultat me semble plutôt heureux.

2 Blues stochastique

2.1 Introduction

Les premières expérimentations consistant à faire composer de la musique par un ordinateur datent des années 50. En 1957, L.A. Hiller et L.M. Isaacson utilisent l'ordinateur ILLIAC 1 de l'Université de l'Illinois pour composer la *« suite Illiac »*, quatuor à cordes en quatre mouvements, chaque mouvement correspondant à une expérimentation différente **[8, 9, 10]**. Cette pièce combine l'utilisation de nombres pseudo-aléatoires et de règles diverses.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Dans le système MIDI, nous disposons de 128 notes numérotées de **0** (*do -2*) à **127** (*sol 8*), sur plus de dix octaves. Nous pourrions bien sûr tirer des notes au hasard parmi ces valeurs, mais l'on imagine assez bien le résultat. Nous allons plutôt tenter de créer quelque chose pouvant être considéré comme une mélodie. Pour rester simple, on peut considérer qu'une mélodie est une suite de notes plus ou moins proches les unes des autres qui montent et qui descendent dans des intervalles assez limités (en général une ou deux octaves). L'idée vient donc d'effectuer une marche au hasard en utilisant une méthode Monte-Carlo : on part d'une note, on tire un nombre au hasard entre **0** et **1**, s'il est supérieur à **0,5** on passe à la note au-dessus, sinon à la note en dessous.

2.2 I've got the blues

Si l'on utilise la numérotation MIDI, nous parcourrons alors des gammes chromatiques : *do*, *do#*, *ré*, *ré#*, *mi*, *fa*, *fa#*, *sol*, *sol#*, *la*, *la#*, *si*, etc. Nous sommes alors dans la musique dite dodécaphonique, développée par Arnold Schönberg au début du XXe siècle, puisque ces douze notes sont sur un pied d'égalité : chacune aura la même probabilité d'être jouée. Nous aurions pu nous lancer dans la musique sérielle, mais nous allons ici rester dans la musique tonale. La première chose que nous allons faire est donc de définir un tableau **gammes[]** qui contiendra un sous-ensemble des 128 notes MIDI composé d'une gamme commençant par la note **0** (c'est un *do*) et répétée aux octaves supérieures. La plupart des gammes couramment utilisées comportent entre cinq et sept notes : ce nombre variable sera stocké dans **nb_notes**. Ligne 22, nous commençons à remplir le tableau avec les notes de notre gamme sur une octave. Il s'agit ici d'une gamme de blues (six notes) : **{0, 3, 5, 6, 7, 10}** soit *do*, *ré#*, *fa*, *fa#*, *sol*, *la#*. La boucle « Faire Tant Que » et sa boucle « Pour » interne sont chargées de répéter cette gamme aux octaves supérieures tout en veillant à ne pas dépasser la note MIDI **127**. La variable **jmax** contiendra l'indice de la dernière note entrée dans le tableau.

```
13: void ecrire_piste2(FILE *fichier) {
14:     double p, delta ;
15:
16:     unsigned long marque = MIDI_ecrire_en_tete_piste(fichier) ;
17:     MIDI_Control_Change(fichier, 0, reverb, 64) ;
18:     MIDI_Control_Change(fichier, 0, 1, 40) ;    // modulation
19:     MIDI_Program_Change(fichier, 0, 30) ;
20:
21:     const int nb_notes = 6 ;
22:     unsigned char gammes[128] = {0, 3, 5, 6, 7, 10} ;  // Blues
23:     int jmax = nb_notes - 1 ;
24:     int octave = 1 ;
25:     bool continuer = true ;
26:     do {
27:         for(int j=0 ; j <= nb_notes-1 ; j = j+1){
28:             if (gammes[j] + octave*12 <= 127) {
29:                 jmax = octave*nb_notes + j ;
30:                 gammes[jmax] = gammes[j] + octave*12 ;
31:             }
32:             else continuer = false ;
33:         }
34:         octave = octave + 1 ;
35:     } while (continuer) ;
```

L'instrument utilisé (**30**) est une guitare avec distorsion. Ligne 18, le *Control Change* **1** correspond à la molette de modulation d'un clavier musical, qui commande par défaut le vibrato **[11]**.

2.3 Marche au hasard et tonique

La deuxième partie de la procédure **ecrire_piste2()** va consister à créer une mélodie en parcourant nos gammes au hasard. La fonction **srand(time(NULL))** permet de modifier la graine du générateur de nombres pseudo-aléatoires avec l'horloge de l'ordinateur afin d'obtenir une mélodie différente à chaque fois.

Nous partons d'un *do* (24e note de notre gamme) de durée **noireblues** (les constantes seront définies plus loin) avec un volume **40**. Nous tirons ensuite un nombre pseudo-aléatoire **p** entre **0** et **1** en utilisant la fonction **rand()** qui renvoie un entier entre **0** et **RAND_MAX**. Le test lignes 46-51 utilise la valeur de **p** pour déterminer si on passe à la note au-dessus ou en dessous dans notre gamme, ou si on rejoue la même note (avec une probabilité **0.1**). On veille bien sûr à ne pas dépasser les limites du tableau **gammes[]**. On tire ensuite un autre nombre pseudo-aléatoire pour déterminer la durée de la note (lignes 53-55), pour donner un peu de rythme. Notre blues comporte en tout deux cents notes (constante **longueur**).

```
37:   srand(time(NULL));
38:   unsigned char duree = noireblues ;
39:   const unsigned char tonique = 24 ;  // do
40:   unsigned char note = tonique ;
41:   for(int i=1 ; i <= longueur ; i = i+1){
42:     Note_unique_avec_duree(fichier, 0, gammes[note], 40, duree) ;
43:
44:     p = rand() / ((double)RAND_MAX) ;
45:     delta = ((gammes[note] - gammes[tonique]) / 12.0) * 0.45 ;
46:     if (p >= 0.55+delta) {
47:         if (note < jmax) note = note + 1 ;
48:     }
49:     else if (p >= 0.1) {
50:         if (note > 0) note = note - 1 ;
51:     }
52:
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
53:      p = rand() / ((double)RAND_MAX) ;
54:t      if (p >= 0.75) duree = noireblues ;
55:      else duree = noireblues / 4 ;
56:   }
57:   MIDI_fin_de_la_piste(fichier) ;
58:   ecrire_taille_finale_piste(fichier, marque) ;
59: }
```

Mais à quoi sert la variable **delta** ? Si nous nous contentions d'un test **if (p>=0.5)**, la marche au hasard pourrait nous éloigner arbitrairement dans les graves ou les aiguës. La valeur absolue de **delta** est d'autant plus grande que nous nous éloignons de notre note de départ, la tonique. Nous créons ainsi une sorte de force de rappel vers notre tonique qui nous empêche de nous en éloigner de plus d'une octave (douze demi-tons).

2.4 Percussions

Nous allons ajouter une piste avec le rythme de blues de la figure 2 **[12]**. Rappelons que MIDI réserve son 10e canal (le **9** si la numérotation commence à **0**) aux percussions, d'où la déclaration dans le fichier **midi. c** de la constante **percu**. Dans ce canal, il n'y a pas d'instrument à définir, chaque percussion correspondant en fait à un numéro de note allant de **35** (grosse caisse) à **81** (triangle) **[4]**, ce qui correspond à un classement de la plus grave à la plus aiguë. Nous n'utilisons pas notre procédure **Note_unique_avec_duree()**car nous allons devoir jouer ici une à trois notes simultanément et donc gérer à la main les événements MIDI *Note On* et *Note Off* pour chaque note, en n'oubliant pas d'inclure un **delta_time** nul à chaque fois. Les percussions utilisées sont : **35** (*Acoustic Bass Drum,* grosse caisse), **38** (*Acoustic Snare,* caisse claire) et **42** (*Closed Hi-Hat,* charleston fermé). Les volumes sonores ont été choisis différents pour chaque type de percussion.

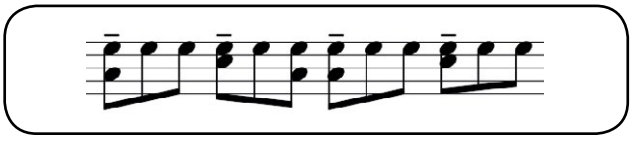


Fig. 2. Un rythme de blues.

Une des caractéristiques des rythmes de blues est l'usage de triolets (groupes de trois notes valant une noire). La durée de chaque coup de charleston sera donc **noireblues/3** (raison pour laquelle il est important que **noireblues** soit un entier multiple de trois). L'opérateur modulo **%** nous est bien utile pour déterminer pour quelles valeurs de **i** il faut taper sur la grosse caisse ou la caisse claire :

```
61: void ecrire_piste_percu(FILE *fichier) {
62:     unsigned long marque = MIDI_ecrire_en_tete_piste(fichier) ;
63:     MIDI_Control_Change(fichier, percu, reverb, 64) ;
64:
65:     for(int i=1 ; i <= longueur*3 ; i = i+1){
66:         MIDI_delta_time(fichier, 0) ;
67:         MIDI_Note(ON, fichier, percu, 42, 80) ; // Charleston
68:         if (i%6 == 4) {                    // Caisse claire
69:             MIDI_delta_time(fichier, 0) ;
70:             MIDI_Note(OFF, fichier, percu, 38, 92) ;
71:             MIDI_delta_time(fichier, 0) ;
72:             MIDI_Note(ON, fichier, percu, 38, 92) ;
73:         }
74:         else if ((i%6 == 1) || (i%12 == 6)) {   // Grosse
caisse
75:             MIDI_delta_time(fichier, 0) ;
76:             MIDI_Note(OFF, fichier, percu, 35, 127) ;
77:             MIDI_delta_time(fichier, 0) ;
78:             MIDI_Note(ON, fichier, percu, 35, 127) ;
79:         }
80:         MIDI_delta_time(fichier, noireblues / 3) ;
81:         MIDI_Note(OFF, fichier, percu, 42, 64) ;
82:     }
83:     MIDI_fin_de_la_piste(fichier) ;
84:     ecrire_taille_finale_piste(fichier, marque) ;
85: }
```

N'oubliez pas de définir les constantes **noireblues** et **longueur** au début du fichier **blues. c** :

```
01: #include "midi. c"
02: #include "time.h"
03: #define noireblues 120
04: #define longueur   200
```

Et de définir le tempo suivant dans la procédure **ecrire_piste1()** :

```
08:     MIDI_tempo(fichier, 1000000) ;
```

Enfin, n'oubliez pas de modifier la fonction **main()** en indiquant qu'il y a trois pistes dans notre fichier MIDI : **ecrire_piste1()**, **ecrire_piste2()** et **ecrire_piste_percu()**.

2.5 Autres explorations

Pour changer d'ambiance, vous pouvez, après avoir désactivé la procédure **ecrire_piste_percu()**, utiliser d'autres gammes **[13]** :

- une gamme par ton, utilisée en particulier par Debussy : **{0, 2, 4, 6, 8, 10}** ;

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

- une gamme pentatonique, pour une ambiance asiatique : `{1, 3, 6, 8, 10}` (les touches noires d'un piano). Utilisez le Koto, instrument `107` ;
- une gamme majeure : `{0, 2, 4, 5, 7, 9, 11}` ;
- une gamme mineure harmonique : `{9, 11, 12, 14, 16, 17, 20, 21}`.

Il faudra également trouver l'instrument le plus en adéquation avec l'ambiance recherchée.

Conclusion et perspectives

Nous avons vu à l'aide de deux exemples comment il était possible d'utiliser le format MIDI pour créer de petites compositions musicales. Dans l'exemple du canon, il s'agissait de répéter des motifs musicaux à l'aide de boucles. Dans l'exemple du blues, nous avons abordé d'une part la musique stochastique et d'autre part l'utilisation des percussions MIDI.

Terminons par quelques idées de travaux pratiques. Le format MIDI faisant correspondre à chaque note un entier, on peut s'amuser à créer des motifs musicaux à partir des codes ASCII de chaînes de caractères, de la même façon que Jean-Sébastien Bach et Dimitri Chostakovitch utilisaient dans leurs œuvres leurs motifs BACH et DSCH en se basant sur la notation musicale anglo-saxonne. On peut également transposer en notes (ou en rythmes) des nombres premiers, des suites mathématiques (Fibonacci, Syracuse, ...), les chiffres de Pi, etc. Et pourquoi pas, s'aventurer en canon dans les rencontres du troisième type : `67`, `69`, `65`, `53`, `60`. ■

Références

[1] MAGNIN V., «Format MIDI : composez en C ! », GNU/Linux Magazine n°196, septembre 2016.

[2] Fichiers musicaux de l'article : http://magnin.plil.net/spip.php?article132

[3] Le canon de Pachelbel : https://fr.wikipedia.org/wiki/Canon_de_Pachelbel

[4] Liste des instruments General MIDI : https://www.midi.org/specifications/item/gm-level-1-sound-set

[5] Numérotation MIDI des notes : https://upload.wikimedia.org/wikipedia/commons/3/36/NoteNamesFrequenciesAndMidiNumbers_V3.svg

[6] Lecteur MIDI Timidity++ : http://timidity.sourceforge.net/

[7] MAGNIN V., « Avec MIDI, lancez-vous dans la musique assistée par ordinateur », Linux Pratique HS n°29, p. 36-47.

[8] FICHET L., « Les théories scientifiques de la musique aux XIXe et XXe siècles », Vrin, 1996, ISBN 978-2-7116-4284-7.

[9] ANDREATTA M., « Musique algorithmique », 2009 : http://articles.ircam.fr/textes/Andreatta11b/index.pdf

[10] Suite Illiac : https://en.wikipedia.org/wiki/Illiac_Suite

[11] Control Change Messages : https://www.midi.org/specifications/item/table-3-control-change-messages-data-bytes-2

[12] Un rythme de blues : http://blog-batteur-debutant.fr/rythme-blues-batterie/

[13] Liste des gammes et modes : https://fr.wikipedia.org/wiki/Liste_des_gammes_et_modes

BlueMind

SOLUTION OPENSOURCE PROFESSIONNELLE DE MESSAGERIE COLLABORATIVE

LIBÉREZ VOTRE MESSAGERIE

3.5

NOUVELLE VERSION

- Emails
- Tâches
- Contacts & Agendas partagés
- Chat
- Mobiles
- Documents & pièces jointes
- Envoi de fichiers volumineux
- Communication unifiée

NOS PROCHAINS ÉVÉNEMENTS

16-17 nov. Paris OpenSource Summit — PARIS PLAINE SAINT-DENIS

novembre Séminaire BlueMind

en savoir plus www.bluemind.net

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

DES MENUS DANS UNE APPLICATION GTK+

Christophe BORELLY [Professeur de l'ENSAM – IUT de Béziers]

Après avoir découvert les rudiments de la programmation GTK+ dans un article précédent, vous allez apprendre maintenant à ajouter des menus à votre application GTK+.

Mots-clés : GTK+, menus, menus contextuels, barre d'outils, mnémoniques, raccourcis clavier, actions

Résumé

Vous vous êtes toujours demandé comment intégrer des menus à une application graphique GTK+ existante, ou comment associer à ces menus des raccourcis clavier ainsi que des icônes ? Vous êtes curieux de savoir comment on peut réaliser un menu contextuel et une barre d'outils ? C'est là tout ce que cet article va vous dévoiler...

Nous avons vu précédemment **[1]** comment créer une petite application GTK+ contenant deux zones de texte, un bouton et permettant de simuler une authentification.

Je vous propose maintenant d'améliorer une application GTK+ un peu plus ludique. Il s'agit d'un jeu de mastermind où le but est de déterminer une combinaison de quatre couleurs en plusieurs essais. Bien sûr la plus grande partie du code est déjà réalisée (mais il n'est pas terminé - voir note). Nous n'allons voir ici que les différentes façons d'ajouter un menu à cette application.

Note

Si vous désirez terminer l'application, et c'est un bon exercice d'algorithmique, il vous faudra compléter la fonction `checkCombinaison()` dans le fichier `lib-mastermind-check.c` que vous pourrez trouver sur le GitHub du magazine. En utilisant les variables globales `code` et `combinaison`, il vous faudra mettre à jour les valeurs `nbBienPlace` et `nbMalPlace` et retourner la valeur `TRUE` lorsque le code a été trouvé.

C'est l'objet d'un des travaux pratiques que je fais faire aux étudiants de première année de DUT Réseaux et Télécommunications, alors pour ne pas donner la solution du TP, je me devais de laisser cette partie inachevée dans cet article.

Dans la première partie de cet article, je vais vous présenter une première technique qui utilise directement les éléments de la librairie GTK+ avec le système classique de gestion des signaux ou événements utilisateur. Nous verrons ensuite comment ajouter des raccourcis clavier (accélérateurs) et des icônes à ces menus.

Puis, dans la deuxième partie, nous découvrirons comment on peut faire apparaître un menu contextuel (ou surgissant) dans l'application.

Nous présenterons ensuite une nouvelle notion importante : la notion d'**action**. Et nous verrons ce que cela change dans la gestion des événements pour une application.

Enfin, nous terminerons en expliquant comment créer une barre d'outils, la version « iconisée » d'un menu classique.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

1 Menus en GTK+

Les menus en GTK+ utilisent principalement les widgets **GtkMenu** et **GtkMenuItem** (ou ses dérivés) et c'est le signal **activate** qui sera associé à ces éléments. Par défaut, les éléments de ces menus ne comportent que du texte auquel on peut associer un caractère mnémonique et un raccourci clavier (**GtkAccelLabel**), mais nous verrons qu'avec un peu plus de code, on peut tout de même ajouter une icône.

1.1 Menus basiques en GTK+

Pour notre application, le menu permettra de démarrer une nouvelle partie, de paramétrer la difficulté du jeu (taille de la combinaison à trouver) et la durée d'une partie ou bien de terminer le programme (voir figure 1).

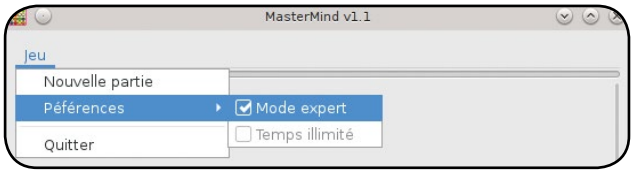


Fig. 1 : Application 1.1 avec un menu à 2 niveaux.

Dans ce premier exemple, le menu est créé dans la fonction **addMenu()** qui est appelée lors de la création de la fenêtre de l'application à la ligne 80. On lui passe en argument la variable **window** car elle est utilisée pour récupérer le contenu de la fenêtre (ligne 37) et par les fonctions de *callback* **newGame()** et **menuQuit()** qui servent à démarrer le jeu ou à arrêter le programme (voir lignes 46 et 55) :

```
...
36 void addMenu(GtkWidget *window) {
37  GtkWidget *content=gtk_bin_get_child(GTK_BIN(window));
38  GtkWidget *menuBar=gtk_menu_bar_new();
39  gtk_container_add(GTK_CONTAINER(content),menuBar);
40  GtkWidget *menuItem=gtk_menu_item_new_with_mnemonic(_("_Jeu"));
41  gtk_menu_shell_append(GTK_MENU_SHELL(menuBar),menuItem);
42  GtkWidget *menuNiveau1=gtk_menu_new();
43  gtk_menu_item_set_submenu(GTK_MENU_ITEM(menuItem),menuNiveau1);
44  menuItem=gtk_menu_item_new_with_mnemonic(_("_Nouvelle partie"));
45  gtk_menu_shell_append(GTK_MENU_SHELL(menuNiveau1),menuItem);
46  g_signal_connect_after(menuItem,"activate",G_
CALLBACK(newGame),window);
47  menuItem=gtk_menu_item_new_with_mnemonic(_("_Péférences"));
48  gtk_menu_shell_append(GTK_MENU_SHELL(menuNiveau1),menuItem);
49  GtkWidget *menuNiveau2=gtk_menu_new(); // Sous menu
50  gtk_menu_item_set_submenu(GTK_MENU_ITEM(menuItem),menuNiveau2);
51  menuItem=gtk_separator_menu_item_new(); // Separateur
52  gtk_menu_shell_append(GTK_MENU_SHELL(menuNiveau1),menuItem);
53  menuItem=gtk_menu_item_new_with_mnemonic(_("_Quitter"));
54  gtk_menu_shell_append(GTK_MENU_SHELL(menuNiveau1),menuItem);
55  g_signal_connect(menuItem,"activate",G_
CALLBACK(menuQuit),window);
56  // Sous menu de préférences
57  menuItem=gtk_check_menu_item_new_with_mnemonic(_("_Mode
expert"));
58  gtk_menu_shell_append(GTK_MENU_SHELL(menuNiveau2),menuItem);
59  g_signal_connect(menuItem,"activate",G_
CALLBACK(menuExpert),NULL);
60  menuItem=gtk_check_menu_item_new_with_mnemonic(_("_Temps
illimité"));
61  gtk_menu_shell_append(GTK_MENU_SHELL(menuNiveau2),menuItem);
62  g_signal_connect(menuItem,"activate",G_CALLBACK(menuTime),NULL);
63  gtk_widget_set_sensitive(menuItem,FALSE); // Désactive ce
menuItem
64 }
...
70 static void startApplication(GtkApplication *app,gpointer data) {
71  loadCSS();
72  GtkWidget *window=gtk_application_window_new(app);
73  gtk_window_set_icon_name(GTK_WINDOW(window),PACKAGE);
...
80  addMenu(window);
...
99  gtk_widget_show_all(window);
100 }
```

La première chose à faire est de créer une barre de menu et de l'ajouter au contenu de la fenêtre (lignes 38 et 39). On crée ensuite un élément de menu (voir le détail dans la suite) que l'on ajoute à cette barre de menu avec la fonction **gtk_menu_shell_append()** à la ligne 41. Il s'agit en fait du niveau 0 de la barre de menu, c'est-à-dire le texte qui est affiché en permanence en haut de la fenêtre en général. Pour ajouter d'autres éléments à cette barre de menu, il faudrait dupliquer les lignes 40 et 41 en changeant le texte bien sûr.

Il existe en fait trois façons différentes pour fabriquer un élément de menu. Soit avec **gtk_menu_item_new()** qui créé un élément de menu vide (sans texte), soit avec **gtk_menu_item_new_with_label()** où l'on indique le texte à afficher, soit enfin avec **gtk_menu_item_new_with_mnemonic()** qui permet en plus de préciser quel caractère sera utilisé comme mnémonique pour ce menu. Le caractère mnémonique est la lettre qui sera soulignée dans le menu et qui pourra l'activer avec la touche **<Alt>**. Par exemple si le caractère mnémonique d'un menu est le « J » (comme à la ligne 40), **<Alt>** + **<J>** active le menu. Au niveau du code, il suffit de faire précéder de _ la lettre que l'on a choisie comme mnémonique. Mais attention à ne pas confondre cela avec la macro **_()** utilisée pour l'internationalisation dans cet exemple !

S'il y a un niveau de menu supplémentaire contenant lui aussi des mnémoniques, actionner une des lettres en question déclenche le sous-menu correspondant. Autrement

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

dit dans notre cas, **<Alt>** + **<J>** + **<N>** lance une nouvelle partie, **<Alt>** + **<J>** + **<P>** + **<M>** active le mode expert et **<Alt>** + **<J>** + **<Q>** arrête le programme.

Un niveau de sous-menu peut être ajouté avec les fonctions **gtk_menu_new()** et **gtk_menu_item_set_submenu()** comme aux lignes 42 et 43. Dans ce sous-menu de niveau 1, on ajoute les éléments de menu « Nouvelle partie », « Préférences », séparateur (voir ligne 51) et « Quitter ». Il faut noter que pour les préférences, un sous-menu de niveau 2 est aussi créé aux lignes 49 et 50. les éléments de ce sous-menu sont ici des **GtkCheckMenuItem** qui permettent d'avoir une case à cocher (voir lignes 57 et 60).

On peut associer une fonction à un élément de menu comme on l'a vu dans l'article précédent pour la gestion des clics sur un bouton, mais ici pour un menu, il faut indiquer le signal **activate**. Dans cet exemple, on a défini une fonction pour chaque action à réaliser (**newGame**, **menuExpert**, **menuTime** et **menuQuit**). Certaines de ces fonctions ont besoin d'interagir avec la fenêtre, c'est pourquoi la variable **window** est donnée en paramètre (voir lignes 46 et 55).

Pour la fonction **menuExpert()** par exemple, on peut récupérer l'état de la case à cocher avec la fonction de la ligne 7. On change alors la taille du code à trouver en modifiant la variable **nbCases** en fonction de l'état.

```
06 static void menuExpert(GtkCheckMenuItem *m,gpointer data) {
07   gboolean state=gtk_check_menu_item_get_active(m);
08   nbCases=(state)?6:4;
09   printf("menuExpert(%d): %d\n",state,nbCases);
10 }
```

Enfin, il faut savoir que l'on peut désactiver un menu en se servant de la fonction **gtk_widget_set_sensitive()** comme l'exemple de la ligne 63 qui désactive le sous-menu « Temps illimité ».

1.2 Menus avec accélérateurs en GTK+

La première évolution que l'on peut apporter à ces menus (voir figure 2), est l'utilisation de raccourcis clavier quelconque (ne commençant pas forcément par **<Alt>**), puis l'écriture de texte avec le langage à balises de **Pango [2]** (la librairie de rendu texte utilisée par GTK+ et **GNOME**).

S'il on veut ajouter des raccourcis clavier, on peut se servir comme à la ligne 48, de la fonction **gtk_widget_add_accelerator()** dans laquelle on indique dans l'ordre : l'élément de menu que l'on veut modifier, le signal associé, le groupe d'accélérateur utilisé dans l'application (créé et ajouté à la fenêtre lignes 37 et 38), le raccourci désiré (la suite de touches en question, ici **<Ctrl>** + **<N>**) et enfin s'il faut afficher le raccourci dans le menu. On peut aussi combiner plusieurs touches en utilisant la barre verticale, par exemple **<Ctrl>** + **<Shift>** s'écrit **GDK_CONTROL_MASK|GDK_SHIFT_MASK**. Notez que le raccourci créé ne pourra pas être modifié logiciellement par l'utilisateur. Nous verrons au paragraphe 1.3, comment il est possible de réaliser cela.

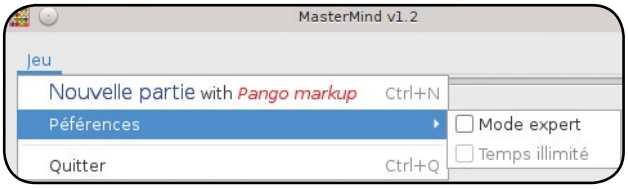


Fig. 2 : Application 1.2 avec un menu amélioré.

Pour l'utilisation du langage à balises de Pango dans le texte du menu, on peut se servir de la fonction **gtk_label_set_markup_with_mnemonic()** comme à la ligne 58 (la fonction **gtk_menu_item_set_label()** ne permet pas l'interprétation des balises). On voit aussi aux lignes 54 et 55 l'utilisation d'une macro définie dans **libmastermind.h** et permettant de créer une chaîne formatée avec deux balises **span**.

À la ligne 57, la fonction **g_markup_printf_escaped()** est utilisée pour formater cette chaîne contenant les balises Pango **[2]**.

```
37  GtkAccelGroup *accelGroup=gtk_accel_group_new();
38  gtk_window_add_accel_group(GTK_WINDOW(window),accelGroup);
...
47   // Ajout d'un raccourci clavier
48   gtk_widget_add_accelerator(menuItem,"activate",accelGroup,
49                              GDK_KEY_n,GDK_CONTROL_MASK,
50                              GTK_ACCEL_VISIBLE);
51   // Utilisation du langage à balises de Pango
52   GtkWidget *child=gtk_bin_get_child(GTK_BIN(menuItem));
53   gtk_label_set_use_underline(GTK_LABEL(child),TRUE);
54   const char *format=SPAN(12.0,blue,%s) " with "
55                      SPAN(10.0,red,<i>Pango markup</i>);
56   const gchar *label=gtk_menu_item_get_label(GTK_MENU_
ITEM(menuItem));
57   char *markup=g_markup_printf_escaped(format,label);
58   gtk_label_set_markup_with_mnemonic(GTK_LABEL(child),markup);
59   g_free(markup);
...
```

1.3 Menus intégrants des icônes en GTK+

Si l'on veut ajouter une petite icône dans un élément de menu (voir figure 3), cela devient un peu plus long à programmer car il faut définir manuellement le contenu du menu (on peut toujours écrire une fonction pour cela d'ailleurs).

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

COMPRENDRE, UTILISER ET ADMINISTRER LINUX

GNU LINUX PRATIQUE ESSENTIEL

À partir du 23 décembre,

Linux Pratique change...

...découvrez sa

Nouvelle formule !

+ de Raspberry Pi

+ de tutoriels

+ d'initiation à la programmation

+ de logithèque

Les nouvelles rubriques de votre magazine :

- Cahier Raspberry Pi & débutant Linux
- Programmation & scripts
- Logithèque & applicatif
- Mobilité & objets connectés
- Système & personnalisation
- Web & réseau
- Terminal & ligne de commandes
- Entreprise & organisation
- Réflexion & société ...

COMPRENEZ, UTILISEZ & ADMINISTREZ LINUX SUR PC, MAC & RASPBERRY PI AVEC LINUX PRATIQUE !

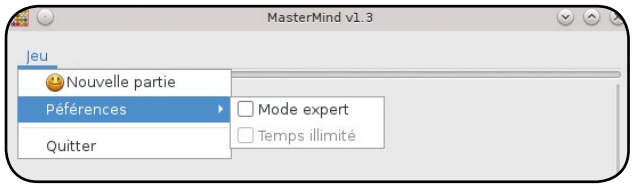


Fig. 3 : Application 1.3 avec un menu comportant une icône.

La première chose à faire est de créer une boite horizontale (ligne 53) pour y placer l'icône et le label (lignes 56 et 58). L'icône est obtenue à partir de la banque de noms internes à GTK+ **[3]** à la ligne 54. Pour le label, on le place aligné à gauche ligne 58 avec la fonction **gtk_box_pack_end()**. On crée ensuite un élément de menu vide (ligne 61), auquel on associe le raccourci clavier à utiliser à la ligne 62.

```
52  // Création d'une boite pour ajouter une icône et un label
53  GtkWidget *menuBox=gtk_box_new(GTK_ORIENTATION_HORIZONTAL,2);
54  GtkWidget *icon=gtk_image_new_from_icon_name("face-smile",
55                                               GTK_ICON_SIZE_MENU);
56  gtk_container_add(GTK_CONTAINER(menuBox),icon);
57  GtkWidget *label=gtk_accel_label_new("_Nouvelle partie");
58  gtk_box_pack_end(GTK_BOX(menuBox),label,TRUE,TRUE,0);
...
61  menuItem=gtk_menu_item_new();
62  gtk_menu_item_set_accel_path(GTK_MENU_
ITEM(menuItem),pathNewGame);
63  gtk_container_add(GTK_CONTAINER(menuItem),menuBox);
```

Cette fois-ci on a utilisé un nom de chemin accélérateur (**pathNewGame**) et la fonction **gtk_menu_item_set_accel_path()** plutôt que la fonction **gtk_widget_add_accelerator()**. L'avantage est que l'on peut par la suite modifier les raccourcis à la demande avec la fonction **gtk_accel_map_change_entry()**.

La création des chemins de cet exemple est réalisée juste après la création du groupe d'accélérateurs. Il faut définir les chaînes représentant les chemins en respectant un format particulier (voir lignes 40 et 41) puis on ajoute des entrées liant un chemin et un raccourci clavier. Ici, **<Ctrl> + <N>** est associé à **pathNewGame** et **<Ctrl> + <Q>** à **pathQuit**.

```
37  GtkAccelGroup *accelGroup=gtk_accel_group_new(); // Groupe
d'accélérateurs
38  gtk_window_add_accel_group(GTK_WINDOW(window),accelGroup);
39  // Création de chaines représentant les accélérateurs
40  const gchar*pathNewGame=g_intern_static_string("<CB-Master>/
newGame");
41  const gchar*pathQuit=g_intern_static_string("<CB-Master>/quit");
42  gtk_accel_map_add_entry(pathNewGame,GDK_KEY_n,GDK_CONTROL_MASK);
43  gtk_accel_map_add_entry(pathQuit,GDK_KEY_q,GDK_CONTROL_MASK);
```

2 Menus contextuels

Si l'on veut avoir la possibilité de créer un menu avec le bouton droit de la souris à l'endroit où l'on a cliqué dans la fenêtre, il va falloir utiliser un conteneur spécial (**GtkEventBox**) et le signal **button-press-event**. On modifie donc un tout petit peu la fonction **startApplication()** qui fabrique la fenêtre et son contenu.

Jusqu'à présent, on ajoutait la zone des combinaisons **vbox** dans la zone de **scroll**, maintenant on va ajouter la **vbox** dans une **ebox** et celle-ci dans la zone de **scroll** (lignes 145 et 146) :

```
...
125  GtkWidget *content=gtk_box_new(GTK_ORIENTATION_VERTICAL,2);
126  gtk_container_add(GTK_CONTAINER(window),content);
127  addMenu(window);
128  // Zone de déclenchement du popup
129  GtkWidget *ebox=gtk_event_box_new();
130  createPopupMenu(window,ebox);
...
145  gtk_container_add(GTK_CONTAINER(ebox),vbox);
146  gtk_container_add(GTK_CONTAINER(scroll),ebox);
...
```

Cela ne change rien au niveau visuel car une **ebox** est transparente par défaut. Ici j'ai ajouté un petit style pour voir directement la zone sensible de la **ebox** (voir figure 4 en page suivante) :

```
GtkEventBox {background-color:#FFEFBC;}/* zone popup - jaune clair */
```

Notez aussi, à la ligne 130, l'utilisation de la fonction **createPopupMenu()** dans laquelle on va créer le menu spécifique (le menu contextuel) et associer, à la ligne 58, le signal **button-press-event** à la fonction **showPopup()** qui affichera ce menu à la demande. C'est pour cela que l'on passe en paramètre de cette fonction, le menu que l'on vient de créer.

```
50 void createPopupMenu(GtkWidget *window,GtkWidget *ebox) {
51  GtkWidget *pMenu=gtk_menu_new();
52  GtkWidget *menuItem=gtk_menu_item_new_with_label(_("Nouvelle
partie"));
53  gtk_menu_shell_append(GTK_MENU_SHELL(pMenu),menuItem);
54  g_signal_connect(menuItem,"activate",G_CALLBACK(newGame),window);
55  menuItem=gtk_menu_item_new_with_label(_("Quitter"));
56  gtk_menu_shell_append(GTK_MENU_SHELL(pMenu),menuItem);
57  g_signal_connect(menuItem,"activate",G_CALLBACK(menuQuit),window);
58  g_signal_connect_swapped(ebox,"button-press-event",
59                           G_CALLBACK(showPopup),pMenu);
60 }
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

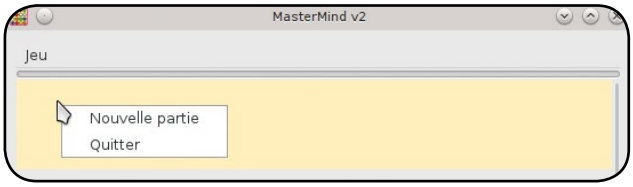


Fig. 4 : Application 2 avec un menu contextuel.

La fonction d'affichage du menu popup suit un prototype particulier qui permet de récupérer le paramètre **event** que l'on utilise pour détecter l'appui sur le bouton de droite de la souris (lignes 7 et 8). Dans ce cas, on appelle la fonction **gtk_menu_popup()** en indiquant le menu à afficher. Les autres paramètres à part les deux derniers ne sont pas utiles ici et sont fixés à **NULL** :

```
06 static int showPopup(GtkMenu *popupMenu,GdkEventButton *event) {
07   if (event->type==GDK_BUTTON_PRESS) {
08     if (event->button==3) { // RIGHT BUTTON
09       gtk_widget_show_all(GTK_WIDGET(popupMenu));
10       gtk_menu_popup(GTK_MENU(popupMenu),NULL,NULL,NULL,NULL,
11                      event->button,event->time);
12       return TRUE;
13     }
14   }
15   return FALSE;
16 }
```

3 Menus utilisant des actions

Une action permet d'associer un nom à une fonctionnalité de l'application, elle est activable et peut avoir des paramètres. Nous verrons également qu'il est assez simple d'y associer un ou plusieurs raccourcis clavier. L'intérêt d'une action est qu'elle peut être déclenchée par plusieurs *widgets*.

Pour utiliser des actions (**GAction**), on peut se servir des éléments **GMenu** et **GMenuItem** qui ne font pas partie de GTK+, mais plutôt de la librairie **GIO [4]**. Voici donc un exemple de création du menu avec cette librairie.

```
96 void addGMenu(GtkApplication *app) {
97   g_set_application_name(_("_Jeu"));
98   GMenu *gMenu=g_menu_new();
99   g_menu_append(gMenu,_("_Nouvelle partie"),"app.newGame");
100  GMenu *prefMenu=g_menu_new();
101  g_menu_append(prefMenu,_("_Mode expert"),"app.modeExpert");
102  g_menu_append(prefMenu,_("_Temps illimité"),"app.
noTime(false)");
103  g_menu_append_submenu(gMenu,_("_Péférences"),G_MENU_
MODEL(prefMenu));
104  g_object_unref(prefMenu);
105  GMenuItem *gMenuItem=g_menu_item_new(_("_Quitter"),"app.quit");
106  GError *error=NULL;
107  GIcon *icon=g_icon_new_for_string("application-exit",&error);
108  if (error!=NULL) {
109    fprintf(stderr,"Unable to load icon : %s !\n",error->message);
110    g_error_free(error);
111  }
112  g_menu_item_set_icon(gMenuItem,icon);
113  g_menu_append_item(gMenu,gMenuItem);
114  g_object_unref(gMenuItem);
115  gtk_application_set_app_menu(app,G_MENU_MODEL(gMenu));
116  g_object_unref(gMenu);
117 }
```

On utilise en argument de la fonction **addGmenu()** la variable **app** car le menu est ajouté à l'application avec la fonction **gtk_application_set_app_menu()** à la ligne 115. Ce qui change par rapport à ce que l'on faisait précédemment, c'est que lorsque l'on ajoute un élément de menu (voir lignes 99, 101, 102 et 105), on indique le nom de l'action qui sera exécutée. La portée de l'action est fixée avec le premier mot : commencer par **app** indique donc que l'action est définie pour toute l'application. Il est possible de définir des actions spécifiquement pour une fenêtre en se servant du préfixe **win**. L'exemple de la ligne 102 montre aussi comment désactiver une action et l'élément de menu correspondant.

Pour le dernier élément de menu (Quitter), je détaille aussi comment ajouter une icône en créant tout d'abord un **GMenuItem** à ligne 105, puis en recherchant une icône à partir de son nom (ligne 107) et enfin en ajoutant l'icône à l'élément de menu (voir ligne 112).

Pour ce qui est des raccourcis clavier, j'ai écris la fonction **addAccels()** détaillant la façon d'associer une action à un ou plusieurs raccourcis (voir lignes 87 et 90). La première chose à faire (ligne 83), est d'enregistrer pour l'application la liste des différentes actions (voir **appEntries** à la ligne 72). On en profite pour fixer le paramètre utilisateur (**data**) qui sera fourni aux fonctions *callback* précisées en deuxième argument des lignes 73 à 76.

Ensuite, à la ligne 86, on crée un tableau de raccourcis (terminé par **NULL**) et à l'aide de la fonction **gtk_application_set_accels_for_action()** sur la ligne suivante, on associe ce tableau à l'action **newGame**. Les raccourcis sont alors automatiquement affichés dans les menus utilisant cette action. Pour l'action **quit**, vous avez un exemple avec deux raccourcis clavier pour la même action (voir lignes 89 à 91).

```
72 static GActionEntry appEntries[]={
73   {"modeExpert",actionExpert,NULL,"false",changeModeExpert},
74   {"noTime",actionTime,NULL,"false",changeTime},
```

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

```
75  {"newGame",actionNewGame,NULL,NULL,NULL},
76  {"quit",actionQuit,NULL,NULL,NULL}
77 };
...
82 void addAccels(GtkApplication *app,gpointer data) {
83  g_action_map_add_action_entries(G_ACTION_MAP(app),
84                                  appEntries,G_N_ELEMENTS(appEntries),
85                                  data);
86  const gchar *newGameAccels[2]={"<Ctrl>N",NULL};
87  gtk_application_set_accels_for_action(GTK_APPLICATION(app),
88                                        "app.newGame",newGameAccels);
89  const gchar *quitAccels[3]={"<Ctrl>Q","<Control><Shift>X",NULL};
90   gtk_application_set_accels_for_action(GTK_APPLICATION(app),
91                                         "app.quit",quitAccels);
92 }
```

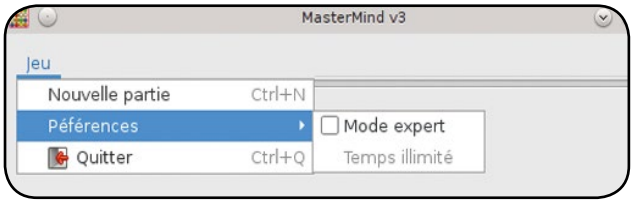


Fig. 5 : Application 3 utilisant des actions.

Il existe néanmoins de petites différences dans la gestion des actions par rapport à l'utilisation des événements et des signaux. On garde ici le même principe de fonction *callback* utilisé classiquement avec les signaux, mais avec un prototype un peu différent (voir lignes 16 et 17). On a cette fois-ci trois arguments : l'action qui a généré l'appel, un paramètre optionnel qui lui est associé et enfin une donnée **data** fournie à toutes les actions et spécifiée à la ligne 85. Dans cet exemple, seulement deux actions utilisent ce paramètre (voir lignes 73 et 74). Il s'agit de la quatrième valeur du tableau de **GActionEntry** fixé à **false** par défaut dans cet exemple. La dernière valeur du tableau spécifie le nom de la fonction à exécuter lorsque ce paramètre est modifié. Par exemple pour l'action **modeExpert** (voir ligne 73), la fonction de modification est **changeModeExpert()**.

Pour cette action donc, on veut pouvoir sauvegarder une valeur booléenne indiquant si le mode expert est activé ou non. Au niveau de la fonction **actionExpert()**, on récupère alors l'ancien état de l'action (de type **GVariant**) à la ligne 18 et 19. Ensuite à la ligne 21, on appelle automatiquement et implicitement la fonction **changeModeExpert()** avec la valeur opposée de **enabled**. Dans cette dernière fonction, on récupère la valeur (ligne 08), on change ensuite la difficulté du jeu (le nombre de cases du code à découvrir) en fonction de cette valeur et on sauvegarde enfin le nouvel état de l'action en question (voir ligne 11).

```
06 static void changeModeExpert(GSimpleAction *action,
07                               GVariant *state,gpointer data) {
08   gboolean enabled=g_variant_get_boolean(state);
09   nbCases=(enabled)?6:4;
10   printf("changeModeExpert(%d): %d\n",enabled,nbCases);
11   g_simple_action_set_state(action,state);
12 }
...
16 static void actionExpert(GSimpleAction *action,
17                           GVariant *parameter,gpointer data) {
18   GVariant *state=g_action_get_state(G_ACTION(action));
19   gboolean enabled=g_variant_get_boolean(state);
20   printf("actionExpert(%d)\n",enabled);
21   g_action_change_state(G_ACTION(action),g_variant_new_
boolean(!enabled));
22   g_variant_unref(state);
23 }
```

4 Barre d'outils

La barre d'outils est une version « iconisée » d'un menu. En GTK+, il suffit d'utiliser la fonction **gtk_toolbar_new()** pour obtenir une barre d'outils. On lui ajoute ensuite des éléments de type **GtkToolItem** (**GtkToolButton**, **GtkToggleToolButton**, **GtkRadioToolButton** ou bien **GtkSeparatorToolItem**). Enfin, on ajoute cette barre d'outils comme un widget classique à la fenêtre désirée.

Fig. 6 : Application 4 avec barre d'outils.

Dans cet exemple, vous pourrez voir les cas les plus courants pour les éléments d'une barre d'outils (voir figure 6). En premier, de la ligne 101 à 106, on ajoute un bouton avec icône et une chaîne de description intégrant le langage à balise de Pango comme en section 1.2. La valeur **-1** utilisée dans la fonction **gtk_toolbar_insert()** à la ligne 105, permet d'ajouter l'élément à la fin de la barre d'outils. Si l'on veut placer un outil en début de barre, il faut mettre **0**. Enfin, l'instruction de la ligne 106, associe l'outil à l'action désirée.

On peut ajouter un séparateur d'outils comme indiqué aux lignes 108 et 109.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Pour le bouton de démarrage d'une partie, on indique à l'application que le label est important et qu'il faut l'afficher en permanence (voir ligne 114).

Enfin, pour la gestion du mode expert de l'application qui sera activé ou non, on se sert d'un **GtkToggleToolButton** pour lequel on indique à la ligne 124, qu'il est désactivé par défaut. La figure 6 montre l'aspect de ce bouton quand il a été activé.

```
96 void addToolBar(GtkWidget *window) {
97  GtkWidget *content=gtk_bin_get_child(GTK_BIN(window));
98  GtkWidget *toolBar=gtk_toolbar_new();
99  gtk_container_add(GTK_CONTAINER(content),toolBar);
100  // Outil pour quitter l'application
101  GtkWidget *icon=gtk_image_new_from_icon_name("application-exit",
102                                               GTK_ICON_SIZE_MENU);
103  GtkToolItem *item=gtk_tool_button_new(icon,_("Quitter"));
104  gtk_tool_item_set_tooltip_markup(item,_("Terminer
l'<b>application</b>"));
105  gtk_toolbar_insert(GTK_TOOLBAR(toolBar),item,-1); // Append
106  gtk_actionable_set_action_name(GTK_ACTIONABLE(item),"app.quit");
107  // Séparateur
108  item=gtk_separator_tool_item_new();
109  gtk_toolbar_insert(GTK_TOOLBAR(toolBar),item,-1);
110  // Outil pour commencer une partie
111  icon=gtk_image_new_from_icon_name("face-smile",
112                                    GTK_ICON_SIZE_MENU);
113  item=gtk_tool_button_new(icon,_("Nouvelle partie"));
114  gtk_tool_item_set_is_important(item,TRUE); // Display label
115  gtk_tool_item_set_tooltip_text(item,_("Démarrer une nouvelle
partie"));
116  gtk_toolbar_insert(GTK_TOOLBAR(toolBar),item,-1);
117  gtk_actionable_set_action_name(GTK_ACTIONABLE(item),"app.
newGame");
118  // Outil pour activer le mode expert
119  item=gtk_toggle_tool_button_new();
120  gtk_tool_button_set_label(GTK_TOOL_BUTTON(item),_("Mode
expert"));
121  gtk_tool_item_set_tooltip_text(item,_("Modifier le mode
expert"));
122  gtk_toolbar_insert(GTK_TOOLBAR(toolBar),item,-1);
123  gtk_actionable_set_action_name(GTK_ACTIONABLE(item),"app.
modeExpert");
124  gtk_toggle_tool_button_set_active(GTK_TOGGLE_TOOL_
BUTTON(item),FALSE);
125 }
```

Conclusion

Voilà, vous connaissez maintenant les différentes façons d'ajouter manuellement un menu à une application GTK+. Mais vous avez dû vous rendre compte que l'écriture peut devenir un peu fastidieuse par moment et que l'on passe beaucoup de temps à écrire du code pour l'organisation et le paramétrage des éléments et pas pour les fonctionnalités réelles de l'application. Il existe bien sûr une solution à ce problème basée sur **GtkBuilder** et un outil de développement rapide d'application (**Glade**). Je vous propose donc de voir tout cela dans un prochain article. ■

Références

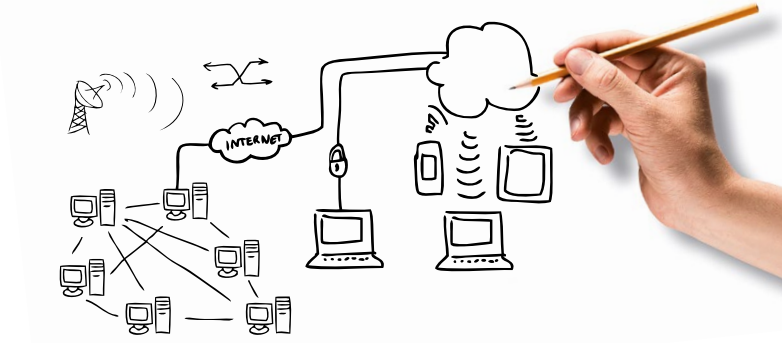
[1] BORELLY C., « Programmer avec GTK+ », GNU/Linux Magazine n°194, juin 2016, p. 58 à 65.

[2] Le langage à balise de Pango : https://developer.gnome.org/pango/stable/PangoMarkupFormat.html

[3] Liste des noms d'icônes standards : https://developer.gnome.org/icon-naming-spec/#names

[4] Page de GIO sur Wikipedia : https://en.wikipedia.org/wiki/GIO_%28software%29

LICENCE PRO
RÉSEAUX ET TÉLÉCOMS



UN BAC+3 QUI
AIME
L'OPEN-SOURCE
À L'IUT DE BÉZIERS

IUT Béziers
iutbeziers.fr/merit

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

AJAX AVEC JQUERY

Laurent NAVARRO [Développeur @Toulouse]

Le Web moderne est dopé à l'AJAX : jQuery peut être un allié de choix pour mettre en œuvre AJAX dans vos pages et je vous invite aujourd'hui à étudier de plus près les diverses façons de le faire à travers jQuery.

Mots-clés : jQuery, HTML, AJAX, JavaScript, Web, Programmation

Résumé

L'objectif de cet article est de rappeler/présenter les diverses approches disponibles dans jQuery pour mettre en œuvre AJAX. Peu de choses vous seront nécessaires, un éditeur de texte, un navigateur Web, et télécharger jQuery ici http://jquery.com/download/ ou directement le fichier utilisé dans les exemples http://code.jquery.com/jquery-2.2.0.js

Pour aller plus loin, la documentation de référence pourra aussi vous être utile [1].

Nous l'avons vu dans un précédent article (GLMF N°194), JQuery est une bibliothèque **JavaScript** de manipulation du DOM (*Document Object Model*) open source très largement répandue. Mais la dernière fois, nous n'avions pas abordé une partie importante qui est l'intégration d'AJAX.

Mais qu'est-ce que c'est qu'AJAX (hormis une gamme de produit d'entretien apparue en 1947, un club de football néerlandais et un personnage de la mythologie grecque) ?

Le terme AJAX, apparu en 2005, est l'acronyme de « *asynchronous JavaScript and XML* » et désigne un ensemble de technologies permettant de faire des communications asynchrones avec un serveur Web et d'adapter le contenu de la page dynamiquement avec la réponse.

Commençons par un peu d'histoire : au début du Web et jusqu'à la fin du siècle dernier, le seul moyen de communiquer avec un serveur Web était de recharger la page. En ce temps-là, les pages étaient plutôt statiques, JavaScript (qui était d'une portabilité très très relative) essayait de les rendre un peu dynamiques avec, au début, des pop-up (puis les `IFrame` et d'autres infâmes bidouilles). Puis DHTML et les prémisses de manipulations du DOM sont apparus. Il faut cependant bien reconnaître que tout ceci était assez chaotique et compliqué à mettre en œuvre, car il y avait déjà des problèmes de version de navigateur.

De nos jours, AJAX désigne surtout la capacité à faire une communication asynchrone avec un serveur depuis du code JavaScript au moyen de l'objet `XMLHttpRequest`.

Avec AJAX, une requête émise est généralement asynchrone et donc non bloquante pour le code qui l'a initiée. Lorsque la réponse arrive, un événement est déclenché pour permettre au développeur de traiter la réponse. Les requêtes peuvent être synchrones/bloquantes, mais on reste sur une logique d'événements.

Nous n'allons traiter dans cet article que l'interface proposée par jQuery, mais sachez qu'il est possible de faire de l'AJAX sans bibliothèque ou avec d'autres bibliothèques.

1 Premières requêtes AJAX avec load

Nous allons commencer par la primitive jQuery de haut niveau qui permet de charger directement du contenu **HTML**. En effet, au début d'AJAX, il était imaginé de principalement utiliser le format **XML** pour communiquer de façon asynchrone, aujourd'hui c'est principalement **JSON** et HTML qui sont utilisés dans les communications AJAX.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

La méthode jQuery **load(url)** va nous permettre de facilement récupérer un fragment de code HTML via une requête HTTP GET et de le mettre dans un élément du DOM comme nous l'avons vu précédemment avec la méthode **html**.

Le code suivant va nous permettre, lors d'un clic sur le bouton de modifier le contenu de la *div* **demo1** pour y mettre le contenu du fichier **HelloWorld.html** :

```
01: <script src="jquery-2.2.0.js"></script>
02: <div id="demo1" style="border: black solid 1px;width: 300px;" >
03:   ICI je veux mettre mon texte dynamique
04: </div>
05: <button onclick="$('#demo1').load('HelloWorld.html');">Hello
World</button>
```

Le fichier **HelloWorld.html** est simplement :

```
<b>Hello</b> World
```

Nous obtenons le résultat présenté en figure 1.

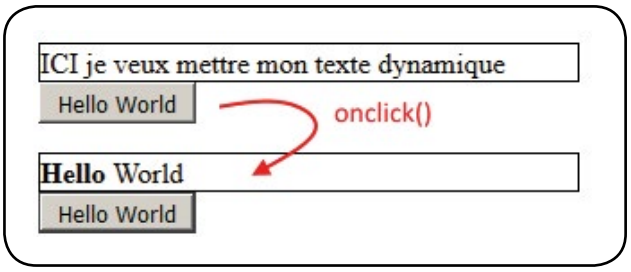


Fig. 1 : Modification de contenu HTML par une requête AJAX load()

Ligne 1 : il faut commencer par inclure jQuery.

Lignes 2 à 4 nous définissons le conteneur cible en lui fixant l'**id demo1**.

Ligne 5, lors du clic sur le bouton, nous invoquons la méthode **load** depuis notre élément **#demo1** en lui passant en paramètre l'URL **HelloWorld.html**.

Comme vous pouvez le constater, faire de l'AJAX avec jQuery c'est quand même assez facile. La méthode **load** travaille de façon asynchrone (ne bloque pas l'exécution d'autres commandes JavaScript et la page reste utilisable même si la requête est longue) et ne remplace le contenu de l'élément que si la requête aboutit sans erreur.

Charger un fichier statique c'est sympa, mais on va quand même souvent plutôt récupérer des choses dynamiques et donc devoir spécifier des paramètres à notre requête.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

DISPONIBLE DÈS LE 11 NOVEMBRE

GNU/LINUX MAGAZINE HORS-SÉRIE n°87

COMPRENDRE LE KERNEL

PLONGEZ AU CŒUR DE VOTRE SYSTÈME GNU/LINUX !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

Nous allons toujours utiliser la méthode **load** mais en lui passant un second paramètre qui sera un objet JavaScript que l'on pourra écrire sous la forme d'un tableau clé/valeur qui sera passé par la méthode POST. À noter que cet objet peut lui-même contenir des objets en tant que valeur.

```
$('#demo1').load("ajax1.php",{nom:"Laurent"});
```

Ici j'ai développé le côté serveur en PHP pour afficher le paramètre **nom** et afficher le contenu de la variable **$_POST** :

```
<?php
  echo "Hello ".$_POST['nom'];
  var_dump($_POST);
```

Ceci nous donne le résultat de la figure 2.

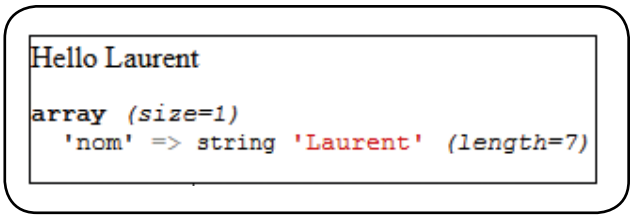


Fig. 2 : Résultat d'une requête AJAX load() paramétrée

Attention : les données transmises par le navigateur sont forcément encodées en UTF-8.

Les données peuvent aussi être passées sous forme de chaînes de caractères encodées pour les URL de ce type **nom=Paul&prenom=Durant**, dans ce cas c'est la méthode GET qui sera utilisée.

2 Utilisation des événements

Plutôt que de charger directement un fragment HTML dans un élément, il est possible d'utiliser des événements pour traiter les données.

Une première approche consiste à ajouter un troisième paramètre à la méthode **load** qui sera invoqué après le remplacement du contenu dans le DOM ou s'il y a une erreur :

```
$('#demo1').load("ajax1.php",{nom:"Laurent"},function(response,
status, xhr){
  if(status=="error")
      $("body").append("Erreur reponse="+response+"<br>");
  else $("body").append("Requete AJAX OK ");
});
```

Si vous préférez vous occuper du chargement dans le DOM vous-même, ou manipuler autre chose que du HTML, vous pouvez plutôt utiliser les deux autres méthodes que sont **$.get** et **$.post** qui ne gèrent que l'approche par événements. Pour ces méthodes, le troisième paramètre est une fonction qui sera invoquée en cas de chargement avec succès (pas en cas d'erreur).

L'équivalent de **$('#demo1').load("ajax1.php",{nom:"Laurent"});** sera

```
$.post("ajax1.php",{nom:"Laurent"},function(response, status, xhr){
  $('#demo1').html(response);
});
```

Vous vous demandez comment nous allons traiter une éventuelle erreur...

Toutes les méthodes AJAX de jQuery, hormis **load**, retournent un objet de type **jqXHR** qui fournit, entre autres, les méthodes **done** et **fail** qui permettent de traiter la bonne exécution ou l'échec.

On pourra faire l'équivalent du **load** ci-dessus avec un traitement d'erreur ainsi :

```
$.post("ajax1.php",{nom:"Laurent"}).done(function(response){
  $('#demo1').html(response);
}).fail(function(response){
  $('#demo1').html("Erreur !!!"+response.responseText);
});
```

J'utilise ici des fonctions anonymes, mais il est, comme toujours, possible d'utiliser des fonctions classiques.

Notez qu'il est possible de s'attacher à des événements de façon globale pour toutes les requêtes AJAX à l'aide des fonctions **$.ajaxSuccess()**, **$.ajaxError()**, etc. Cela peut, par exemple, être utile pour gérer une animation signalant qu'il y a une communication en cours ou un traitement générique des erreurs.

3 Et si on échangeait du JSON

Pour l'instant, nous avons récupéré de la part du serveur des fragments HTML, ce qui convient très bien à de nombreuses situations, mais parfois on souhaite juste récupérer des données et les présenter de façon spécifique avec du JavaScript exécuté sur le client. Il est possible d'échanger les données avec n'importe quel format, cependant des facilités sont à notre disposition si l'on utilise le format JSON. Au début d'AJAX, le format XML était privilégié, mais aujourd'hui c'est le format JSON qui est devenu le plus utilisé.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

JSON signifie « *JavaScript Object Notation* », c'est un format qui permet de représenter des données sous forme structurée et gère les types suivants : chaîne de caractères, numériques, booléens, null, tableaux et associations clés/valeur. Il est directement dérivé de la syntaxe d'initialisation des objets JavaScript.

L'avantage de ce format est qu'il est moins verbeux que XML et naturellement transformable en objets JavaScript.

La méthode **getJSON** va nous permettre de faire une requête AJAX et de récupérer le résultat dans un objet JavaScript directement manipulable.

Dans cet exemple nous allons demander au serveur de nous retourner le nom et le prénom correspondant à l'**ID** numéro **5** :

```
$.getJSON("ajax2.php",{ID:5},function(data){
  $('#demo1').text("Nom = "+data.nom+" ,Prénom = "+data.prenom);
});
```

Côté serveur nous avons le code de test très basique suivant :

```
<?php
  $data=array("nom"=>"Nom ".$_GET['ID'],"prenom"=>"Prénom ".$_
GET['ID']);
  echo json_encode($data);
```

Ce code fabrique un tableau PHP avec les clés **nom** et **prenom** en y mettant des valeurs liées à l'**ID** et le retourne encodé en format JSON grâce à la fonction PHP **json_encode**. Voilà la valeur interceptée par la console réseau de **Firefox** :

```
{"nom":"Nom 5","prenom":"Pr\u00e9nom 5"}
```

La console réseau du menu développeur de Firefox vous aidera beaucoup dans la mise au point de vos requêtes JavaScript car elle vous permettra de voir ce que vous avez envoyé et récupéré, y compris si ce n'est pas du JSON, car le serveur a, par exemple, affiché un message d'erreur.

Côté JavaScript, nous remarquons que nous manipulons directement les attributs **nom** et **prenom** de l'objet **data**. Nous obtenons le texte suivant dans la **div demo1** : **Nom = Nom 5 ,Prénom = Prénom 5**.

Dans le même esprit de spécialisation des primitives, il existe la méthode **getScript** qui permet de charger un fichier JavaScript et de l'exécuter. Ce n'est pas la seule façon d'exécuter du JavaScript, si lors d'un **load** il y a un tag **<script>** dans le HTML il sera aussi exécuté.

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

ACTUELLEMENT DISPONIBLE
LINUX PRATIQUE n°98

UN ORDINATEUR, UN SMARTPHONE OU UN APPAREIL PHOTO CASSÉ ?

SAUVEZ VOS DONNÉES !

NE LE MANQUEZ PAS
CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :
http://www.ed-diamond.com

4 AJAX et les formulaires

De base, en HTML, les formulaires provoquent un changement de page. Vous pouvez convertir vos formulaires en appel AJAX à l'aide des routines précédentes, mais il y existe des facilités pour vous éviter de recopier tous les champs manuellement dans les données de votre appel.

La méthode **serializeArray**, appliquée à un sélecteur vers un formulaire, va permettre de créer un objet prêt à être utilisé avec les méthodes AJAX précédemment vues.

Voilà un exemple typique :

```
<form onsubmit="event.preventDefault(); $('#demo1').load('ajax4.
php',$(this).serializeArray());">
  Nom : <input name="nom"> , Prénom : <input name="prenom">
  <button type="submit">Submit AJAX</button><br>
</form>
```

Nous interceptons l'événement **onsubmit** et bloquons son comportement par défaut à l'aide de l'instruction **event.preventDefault();**. Ensuite, nous sérialisons le contenu de notre formulaire à l'aide de **$(this).serializeArray()**, puis le passons en paramètre à notre requête AJAX réalisée avec la méthode **load** qui affiche le résultat dans **#demo1**.

La méthode **serialize()** fait la même chose, mais retourne une chaîne encodée pour être passée dans une URL. Son utilisation avec la méthode **load**, provoque un appel AJAX similaire, mais qui utilisera la méthode GET au lieu de POST. Il faut généralement préférer la méthode POST qui n'est pas soumise aux restrictions de taille maxi d'une URL et laisse un peu moins de traces sur les paramètres passés.

5 Aller plus loin avec la méthode ajax

Toutes les méthodes que nous avons vues, s'appuient sur la méthode **$.ajax**.

Si vous souhaitez pouvoir manipuler des éléments de détails de la requête (cache, timeout, password,etc.), vous pouvez vous appuyer sur la méthode **ajax**. Cette méthode a tellement de paramètres possibles, qu'il faut les passer à travers un objet. Pour traiter la réponse on utilisera les méthodes **done** et **fail** précédemment étudiées.

L'équivalent de **$('#demo1').load("ajax1.php",{nom:"Laurent"});** avec un traitement d'erreur sera :

```
$.ajax( {
  method: "POST",
  url: "ajax1.php",
  data: { nom:"Laurent" }
}).done(function(response){
  $('#demo1').html(response);
}).fail(function(response){
  $('#demo1').html("Erreur !!!"+response.responseText);
});
```

La méthode **ajax** permettra, entre autres, de faire des requêtes synchrones et donc bloquantes. Bien que cela ne soit pas recommandé par les bonnes pratiques, cela peut avoir du sens pour éviter de devoir gérer des états transitoires. Dans ce cas, il faudra utiliser les paramètres **success** et **error** pour les événements et spécifier **async:false** :

```
$.ajax( {
  method: "POST",
  async:false,
  url: "ajax1.php",
  data: { nom:"Laurent" },
  success: function(response){
    console.log("Reponse reçue");
    $('#demo1').html(response);},
  error: function(response){
  $('#demo1').html("Erreur !!!"+response.responseText); }
});
```

6 Considérations autour d'AJAX

AJAX permet de faire des choses de façon masquées pour l'utilisateur. Les requêtes AJAX sont soumises à la limitation *« Same-origin policy »* qui impose qu'une requête AJAX ne puisse être réalisée que sur le même serveur, le même port et le même protocole que la page initiale. Si vous avez vraiment besoin de faire des requêtes AJAX *cross-domain*, il existe cependant des solutions : la première est l'utilisation de *header* HTTP **Access-Control-Allow-Origin**, l'autre est l'utilisation de **JSONP**, mais nous sortons là de l'utilisation commune d'AJAX et du périmètre de cet article.

L'autre point que je souhaite mettre en lumière est que trop d'AJAX tue l'AJAX. J'ai vu des pages pour lesquelles le trop grand nombre de requêtes AJAX nuisait à l'ergonomie. Ayez en tête que chaque requête AJAX va provoquer une requête HTTP qui va devoir transiter par le réseau et être traitée par le serveur, ce qui peut prendre un certain temps. Il faudra veiller à trouver une architecture qui fait un nombre de requêtes adapté au service rendu.

Un exemple que j'ai en tête, affichait le synoptique d'un bâtiment avec 50 capteurs, et récupérait les données de

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

chaque capteur par une requête AJAX qui nécessitait plus de 200 ms. Du coup il fallait plus de 10 secondes pour l'afficher alors que tout aurait pu être transmis en une seule requête AJAX, voire dans la réponse initiale de la page.

7 Et pour quelques fonctions de plus

jQuery contient quelques fonctions utilitaires que je vais brièvement vous présenter. Ces fonctions sont dans le même esprit que le reste de jQuery, c'est à dire encapsuler un comportement disponible par ailleurs pour qu'il soit disponible simplement de façon indépendante du navigateur. Nous allons voir que certaines de ces fonctions ont un équivalent JavaScript sur la plupart des navigateurs modernes, mais pas forcément sur les navigateurs plus anciens. Au fil du temps, ces fonctions deviennent donc moins utiles, mais si vous avez une audience large en ce qui concerne les navigateurs, elles peuvent encore rendre service. Il est aussi utile de les connaître dans une logique de maintenance de code.

Un premier ensemble de fonctions est autour de la manipulation des collections.

`$.each(Collect, Fct)` permet d'itérer sur tous les éléments de **`Collect`** et d'appeler **`Fct`** pour chaque élément en passant la clé et la valeur en paramètre :

```
var obj1={cle1:"Valeur1",cle2:50};
$.each(obj1, function (cle, val) {
  console.log("cle = " + cle + " ,val=" + val);
});
```

Ce code donnera le résultat suivant :

```
cle = cle1 ,val=Valeur1
cle = cle2 ,val=50
```

`Collect` peut être un tableau ou un objet. Si le *callback* retourne **`false`**, c'est équivalent à un **`break`** dans un **`for`**, ça arrête le parcours de la collection :

```
var Tbl1=[10,20,30,40,50];
$.each(Tbl1, function (cle, val) {
  console.log("cle = " + cle + " ,val=" + val);
  if(val==30) return false; // Sortie anticipée
});
```

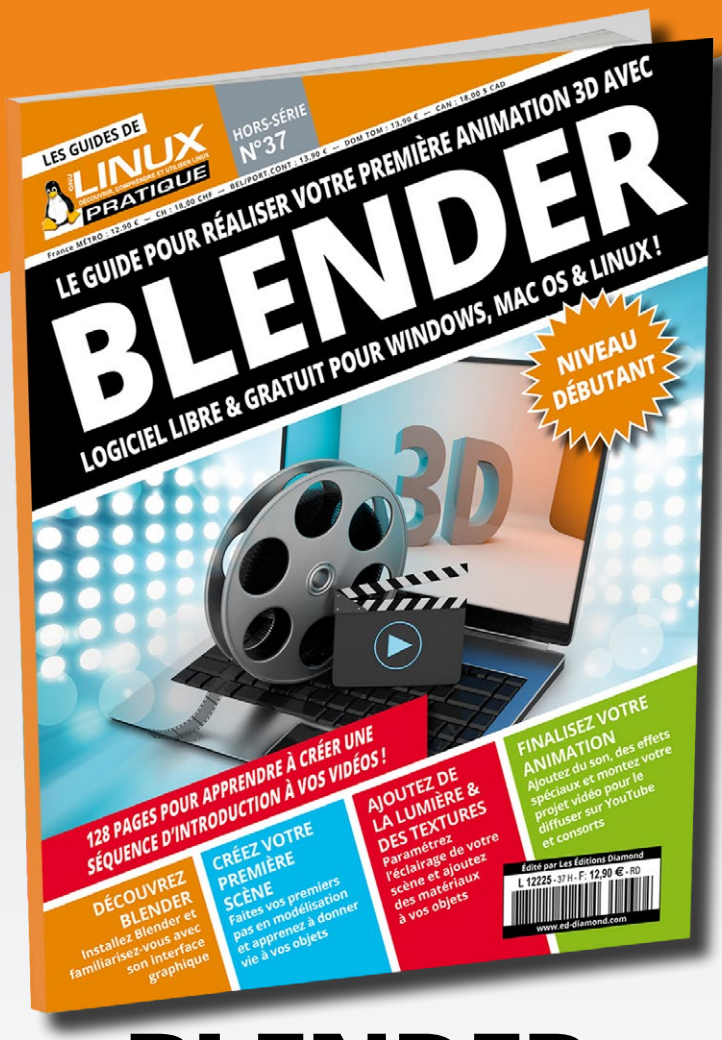
On peut faire un parcours similaire en JavaScript natif ainsi :

```
for (var cle in Tbl1)
   console.log("cle = "+cle+" ,val="+Tbl1[cle]);
```


Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

ACTUELLEMENT DISPONIBLE

LINUX PRATIQUE HORS-SÉRIE n°37



BLENDER

LE GUIDE POUR RÉALISER VOTRE PREMIÈRE ANIMATION 3D !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

http://www.ed-diamond.com

$.grep comme son nom le laisse penser, permet de filtrer les valeurs d'un tableau et les retourne dans un nouveau tableau. Seules les valeurs pour lesquelles la fonction passée en paramètre retourne **true** sont conservées. Voilà un exemple qui retourne les multiples de 3 :

```
var t1=$.grep(Tbl1, function (val) {
  return ((val%3)==0) ;
});
console.log(t1);
```

Ce code qui affiche : **[30]**

$.map permet d'appliquer une fonction à tous les éléments d'un tableau et de retourner une copie transformée du tableau. Ceci est assez similaire à **Array.map**, mais ce dernier ne fonctionne pas, entre autres, sous **IE8**. Cette fonction permet en plus de supprimer des éléments en retournant **null** ou d'en ajouter en retournant un tableau au lieu d'une valeur scalaire :

```
var t1=$.map(Tbl1, function (val) {
  if(val==20) return null;
  if(val==30) return [31,32];
  return 3*val;
});
console.log(t1);
```

Cet exemple nous donnera : **[30,31,32,120,150]**

$.merge permet de modifier un tableau en y ajoutant les valeurs d'un second tableau :

```
var Tbl1=[10,20,30,40,50];
var Tbl2=[60,70];
$.merge(Tbl1, Tbl2);
```

Tbl1 vaut à présent **[10,20,30,40,50,60,70]**.

C'est équivalent au code JavaScript suivant (en plus simple à écrire) :

```
Array.prototype.push.apply(Tbl1, Tbl2);
```

$.extend est l'équivalent de **$.merge** pour les objets :

```
var o1=$.extend({a:1,b:2},{c:3,b:4});
```

o1 contient **{ a: 1, b: 4, c: 3 }**.

Comme **$.merge**, elle modifie le premier objet passé en paramètre et le retourne.

Les utilitaires contiennent aussi quelques fonctions relatives à l'introspection : **isArray**, **isFunction** ,**isPlainObject** ainsi qu'une fonction **$.trim** équivalente à **String.trim** de JavaScript qui efface les caractères invisibles aux extrémités d'une chaîne de caractères, mais qui marche aussi avec **IE8**.

Une autre série de méthodes va nous aider à manipuler les positions et dimensions des objets issus d'un sélecteur.

.position nous donne la position calculée d'un objet et les méthodes **innerHeight**, **height**, **outerHeight**, **width**, etc. permettent de récupérer les dimensions calculées des objets.

Voilà un exemple qui affiche un *div* juste à côté d'une icône **FontAwesome [2]** pour faire un *ToolTip* maison basique. On positionne tout simplement un *div* en positionnement absolu à côté de l'icône quand on passe au-dessus et le masquons quand la souris en sort :

```
Saisir votre nom
  <i class="fa fa-info-circle" style="cursor: pointer;"
     onmouseover="ShowToolTip(this);"
     onmouseout="$('#Div1').hide()"></i>
<div id="Div1" style="border:1px solid black ;width:
150px;position: absolute; display: none;background-color: white;">
  Vous Devez saisir<br>
  Votre Nom </div>
<script>
function ShowToolTip(obj){
  var pos=$(obj).position();
  $("#Div1").css('top',pos.top).css('left',pos.left+$(obj).
outerWidth()).show();
}
</script>
```

Ce qui donne le contenu de la figure 3 lorsque la souris passe sur le « (i) ».

Fig. 3 : Rendu de notre tooltip maison

Conclusion

Voilà, nous avons vu l'essentiel sur l'utilisation d'AJAX avec jQuery, comme vous avez pu le constater, c'est finalement assez simple à mettre en œuvre avec jQuery

J'espère avoir contribué à éclaircir les choses concernant AJAX et que vous intégrerez ces techniques dans vos prochaines applications, si ce n'est déjà fait.

Les fonctions utilitaires, sans être indispensables, peuvent rendre des services et méritent d'être connues. ■

Références

[1] Documentation de jQuery http://api.jquery.com/

[2] Librairie Font-Awesome : https://fortawesome.github.io/Font-Awesome/

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

SERVEURS DÉDIÉS EN PROMOTION
QUANTITÉ LIMITÉE* À PRIX EXCEPTIONNEL

ikoula HÉBERGEUR CLOUD

OFFRES SANS ENGAGEMENT DE DURÉE

SERVEURS	CPU	PROCESSEUR	RAM	DISQUE DUR	SETUP (3)	PRIX HT/MOIS
A LA UNE Green G460	Intel® Celeron® G460 HT	1 CPU (1C/2T) @1,8 GHz	8 Go DDR3	2 To SATA (2)	OFFERT	~~39,99 €~~ 12,99 €
A LA UNE Crazy Fish	Intel® Xeon® 3000	1 CPU (2C/2T) @1,86 GHz min.	4 Go DDR2	2 To SATA (5 400 tr/min)	OFFERT	~~29,99 €~~ 14,99 €
TOP VENTE IKX-G620	Intel® Pentium® G620	1 CPU (2C/2T) @2,6 GHz	8 Go DDR3	1 To SATA	19 €	~~39,99 €~~ 9,99 €
TOP VENTE IKX-Core i3	Intel® Core™ i3 2100T HT	1 CPU (2C/4T) @2,5 GHz	8 Go DDR3	1 To SATA (2)	19 €	~~69,99 €~~ 12,99 €
TOP VENTE IKX-Core i5	Intel® Core™ i5 2400S	1 CPU (4C/4T) @2,5 GHz	16 Go DDR3	1 To SATA (2)	19 €	~~69,99 €~~ 17,99 €
IKX-Core i7	Intel® Core™ i7 2600	1 CPU (4C/8T) @3,4 GHz	16 Go DDR3	1 To SATA (2)	19 €	22,99 €
IKX-3430	Intel® Xeon® Quad Core 3430	1 CPU (4C/4T) @2,4 GHz	8 Go DDR3 (1)	2 x 1 To SATA (2)	39 €	~~189,99 €~~ 25,99 €
Green i5	Intel® Core™ i5 Quad Core 3450	1 CPU (4C/4T) @3,1 GHz	32 Go DDR3	2 To SATA (2)	29 €	25,99 €
TOP VENTE Green i7	Intel® Core™ i7 Quad Core 3770 HT	1 CPU (4C/8T) @3,4 GHz	32 Go DDR3	2 To SATA (2)	29 €	34,99 €
IKX-1220L	Intel® Xeon® E3-1220L HT	1 CPU (2C/4T) @2,2 GHz	32 Go DDR3	2 x 1 To SATA (2)	39 €	~~129,99 €~~ 34,99 €
IKX-R410	Intel® Bi-Xeon® Quad Core 5000	2 CPU (4C/8T) @2 GHz	32 Go DDR3	4 x 1 To SATA Raid 1 Hard (2)	49 €	~~429,99 €~~ 109,99 €
IKX-R710	Intel® Bi-Xeon® Quad Core E5520 HT	2 CPU (4C/8T) @2,26 GHz	32 Go DDR3 (1)	6 x 1 To SATA Raid 1 Hard (2)	49 €	~~499,99 €~~ 189,99 €
IKX-R910	Intel® Quad-Xeon® Hexa Core E7540 HT	4 CPU (6C/12T) @4 GHz	64 Go DDR3 (1)	6 x 300 Go SAS Raid 1 Hard 2,5 (2)	49 €	~~569,99 €~~ 249,99 €

SYSTÈMES LINUX : debian CentOS ubuntu

*Serveurs dédiés disponibles en quantité limitée et sous réserve de disponibilité sur le site : express.ikoula.com/serveur-dedie#promo

(1) Possibilité d'augmenter le niveau de RAM.

(2) Possibilité d'augmenter la taille du / des disque(s) ou d'avoir une alternative SSD / SAS et RAID HARD.

(3) Frais de setup OFFERTS dans le cas d'un engagement annuel non cumulable avec les promotions en cours.

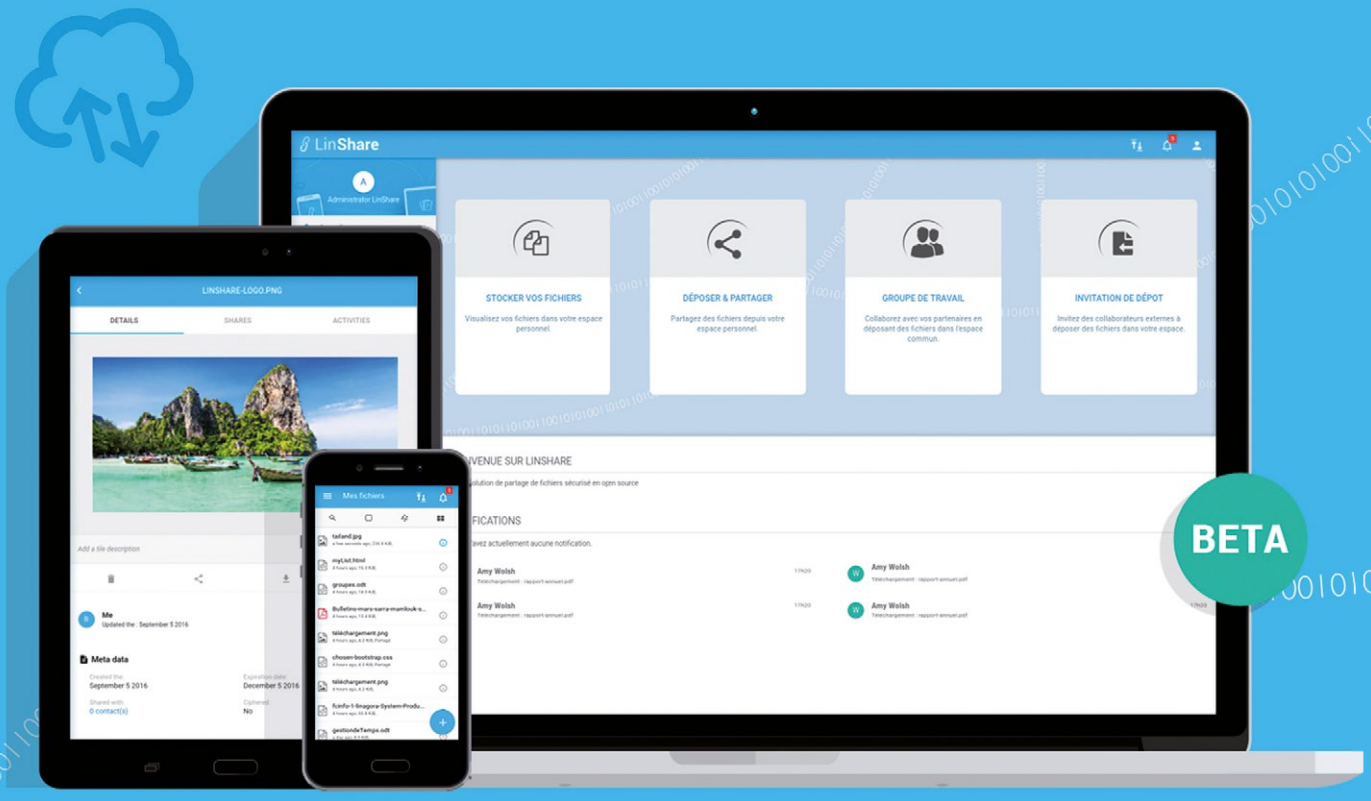
Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

TOUTES LES PROMOTIONS SUR : EXPRESS.IKOULA.COM

ikoula HÉBERGEUR CLOUD
sales@ikoula.com
01 84 01 02 50

LinShare

Partagez **simplement** vos fichiers **en toute sécurité**



TESTEZ LA NOUVELLE VERSION 2.0

www.linshare.org

LINAGORA

Les logiciels libres

www.linagora.fr

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)

Ce document est la propriété exclusive de Johann Locatelli(johann.locatelli@businessdecision.com)