

# Projet UNIX



Générateur de dictionnaire  
d'empreintes multitâche par  
brute-force

Réalisé par :  
Djénéba Djikiné  
Julien Lafont  
Alexandre Bernard  
Christophe Nau

EPSI – CS112

18/04/2011

## TABLE DES MATIÈRES

- I. Analyse du besoin**
  - 1.1. Définition du concept de hachage
  - 1.2. Les empreintes dans le domaine de la sécurité
  - 1.3. Présentation du projet
  - 1.4. Concepts mis en œuvres
- II. Gestion de projet**
  - 1.1. Présentation de Scrum, une pratique sportive
  - 1.2. Estimation et Planification des User-Story
- III. Conception préliminaire**
  - 1.1. Définition des cas d'utilisations
  - 1.2. Architecture logicielle
  - 1.3. Environnement de travail
- IV. Documentation utilisateur**
  - 1.1. Prérequis
  - 1.2. Génération des dictionnaires
  - 1.3. Recherche de résultats
- V. Assurance qualité**
  - 1.1. Conventions de codage
  - 1.2. Librairie de tests unitaires CppTest
  - 1.3. Méthodologie Test Driven Development
- VI. Résultats**
  - 1.1. Impacts de la programmation multitâche
  - 1.2. Amélioration possibles
- VII. Conclusion**

## 1. ANALYSE DU BESOIN





### 1.1. Définition du concept de hachage

On nomme **fonction de hachage** une fonction particulière qui, à partir d'une donnée fournie en entrée, calcule une empreinte servant à identifier rapidement la donnée initiale. Les fonctions de hachage sont utilisées en informatique et en cryptographie.

Les fonctions de hachage servent à rendre plus rapide l'identification des données : calculer l'empreinte d'une donnée ne doit coûter qu'un temps négligeable. Une fonction de hachage doit par ailleurs éviter autant que possible les collisions (états dans lesquels des données différentes ont une empreinte identique).

En cryptographie les contraintes sont plus exigeantes et la taille des empreintes est généralement bien plus longue que celle des données initiales ; un mot de passe dépasse rarement une longueur de 8 caractères, mais son empreinte peut atteindre une longueur de plus de 100 caractères. La priorité principale est de protéger l'empreinte contre une attaque par force brute, le temps de calcul de l'empreinte passant au second plan.

En informatique, les empreintes sont aussi utilisées pour vérifier l'intégrité d'un fichier. Il est courant de trouver à côté d'une image CD son empreinte. Cela permet, une fois le téléchargement terminé, de vérifier si le fichier est toujours valide, en comparant l'empreinte initiale et l'empreinte recalculée du fichier téléchargé.

	<a href="#">debian-6.0.1-i386-CD-1.iso</a>	f81084d7421a8e9b7c3b9969f15f12e2
	<a href="#">debian-6.0.1-i386-CD-2.iso</a>	04e9d25700c56e730334f9d16a877dc6
	<a href="#">debian-6.0.1-i386-CD-3.iso</a>	53efc0a9cd49ed07ba535b625c8de9f3
	<a href="#">debian-6.0.1-i386-CD-4.iso</a>	930000ba7e74c777eb87122147ee21bd

Plusieurs algorithmes peuvent être utilisés pour calculer l'empreinte d'un fichier ou d'une chaîne de caractère, tels que le md5, le sha. Ils sont généralement disponibles en plusieurs déclinaisons en fonction de la taille de l'empreinte générée (sha256, sha512 qui codent sur respectivement 256 et 512bits).

### 1.2. Les empreintes dans le domaine de la sécurité

Dans le domaine de la sécurité, et plus particulièrement sur les sites internet, les fonctions de hachage sont couramment utilisées pour « sécuriser » l'enregistrement des mots de passe. Ceux-ci ne sont plus stockés en clair dans la base de données, seule leur empreinte est conservée.

Cette technique a pour principal objectif de protéger les utilisateurs en cas de piratage de la base de données, les hackers ne pourront pas retrouver le mot de passe à partir de son empreinte (en théorie du moins).

Par la suite, lorsqu'un utilisateur tente de se connecter, on va alors appliquer la même fonction de hachage au mot de passe entré, et le comparer avec celui enregistré.

**Cependant, nous allons voir que les méthodes de hachage sont très loin d'offrir une protection suffisante.**

Tout d'abord, notons que si une fonction de hachage génère par définition une empreinte unique et constante, rien ne garantit que 2 données n'aient pas la même empreinte. Plus l'algorithme de hachage génère une empreinte complexe (en termes de taille notamment), moins ce risque est élevé.

Dans un second temps, plusieurs algorithmes de hachages ont été « cassés » par des hackers, permettant ainsi de retrouver directement, ou plus facilement, le mot de passe initial. Cela est parfois dû au fonctionnement cyclique des algorithmes, ce qui permet d'exploiter une Table arc-en-ciel ([Rainbow Table](#)<sup>1</sup>).

Parmi les algorithmes qui ne sont [plus considérés comme sûrs](#)<sup>2</sup>, on retrouve notamment les premières versions des fonctions MD# et SH# : MD4 (1990), SHA-0 (1993), SHA-1 (1995).

---

## LES ATTAQUES PAR BRUTE-FORCE

Nous avons donc vu que certains algorithmes possèdent de réelles failles structurelles permettant leur décryptage. Mais considérons maintenant les algorithmes qui sont considérés comme sûr (déclinaisons du SHA-2 par exemple).

La question que l'on se pose naturellement est : **sont-ils vraiment inviolables ?**

Si l'on part du postulat que nous n'avons aucune contrainte de temps et de ressources, la réponse est simple : **non**.

Pour cela, il faut utiliser une attaque par **force brute**. Il s'agit de tester, une à une, toutes les combinaisons possibles, et de continuer tant que le hash généré ne correspond pas à celui recherché. Plus le mot de passe initial est complexe (taille, nombre d'alphabets utilisés), plus la recherche pourra être longue. Par ailleurs, certains algorithmes sont plus ou moins rapides pour calculer une empreinte, et la vitesse de calcul est primordiale dans ce genre d'opérations.

---

<sup>1</sup> Tables-en-ciel : [http://en.wikipedia.org/wiki/Rainbow\\_table](http://en.wikipedia.org/wiki/Rainbow_table)

<sup>2</sup> Historique des algorithmes cassés : <http://www.securiteinfo.com/cryptographie/cracked.shtml>

Pendant plusieurs années, les « bonnes pratiques du développement web » incitaient les développeurs à protéger leur mot de passe en les hachant en MD5 (principalement en raison de son intégration native aux SGBDR et aux langages de programmation PHP/JAVA).

*Aujourd'hui encore, l'API PHP ne propose par défaut que les encodages MD5 et SHA-1 qui sont considérés comme peu sécurisés.*

La fonction de hachage md5 permet de générer une empreinte de 32 bits, en voici quelques exemples :

```
« a » => 0cc175b9c0f1b6a831c399e269772661
« motdepasse » => b6edd10559b20cb0a3ddaeb15e5267cc
« 1q58s46dqs86$à@qsd213!.;qsdz » => 8ba3850f135014dfbfa168b76e777e4e
```

En raison de sa popularité dans le cercle des développeurs, plusieurs utilitaires ont vu le jour, permettant de retrouver un mot de passe à partir de son empreinte. Ils entrent dans 2 catégories :

#### LES LOGICIELS DE CASSAGE PAR BRUTE-FORCE

Il utilise le principe développé ci-dessus, en testant pour chaque empreinte à décoder l'ensemble des combinaisons possibles. Ces logiciels fournissent aujourd'hui une puissance de calcul phénoménale grâce à l'utilisation des GPU pour accélérer les calculs (via l'architecture CUDA par exemple).

A titre d'exemple, une machine « grand public » très haut de gamme peut calculer **jusqu'à 3.6 milliards de combinaisons par seconde**. Ce résultat peut-être décuplé en utilisant une ferme de serveurS (botnet ou utilisation d'un service de cloud)

```
Command Prompt - BarsWF_CUDA_x64.exe -h 1b0e9fd3086d90a159a1d6cb86f11b4c -c 0...

BarsWF MD5 bruteforcer v0.8      http://3.14.by/en/md5
by Svarychevski Michail         http://3.14.by/ru/md5

GPU0:  440.91 MHash/sec    CPU0:   34.38 MHash/sec
GPU1:  440.91 MHash/sec    CPU1:   30.18 MHash/sec
GPU2:  440.95 MHash/sec    CPU2:   34.42 MHash/sec
GPU3:  440.91 MHash/sec    CPU3:   34.33 MHash/sec
GPU4:  440.92 MHash/sec
GPU5:  441.48 MHash/sec
GPU6:  440.92 MHash/sec
GPU7:  440.88 MHash/sec

GPU*:  3527.88 MHash/sec    CPU*:  133.30 MHash/sec

Key: t` kN50                Avg.Total: 3611.81 MHash/sec
Hash:1b0e9fd3086d90a159a1d6cb86f11b4c
Progress:  0.90 % ETC      0 days 2 hours 52 min 30 sec
```

Un mot de passe de 8 caractères alphanumériques sera ainsi décodé en 13 minutes (3 821 milliards de combinaisons).

L'inconvénient de cette méthode est que pour chaque empreinte à calculer, le logiciel doit repartir de 0.

---

## LES DICTIONNAIRES D'EMPREINTES

C'est en partant de ce constat que les **dictionnaires d'empreintes** ont vu le jour sur internet. Il s'agit d'immenses bases de données qui sont constamment remplies de nouvelles empreintes. Le pirate n'a qu'à rentrer son empreinte, et si elle a déjà été calculée une fois, il obtiendra le résultat immédiatement.

Les bases de données les plus complètes sont privées, réservées à des cercles de hackers, mais plusieurs sont publiques, avec un nombre d'empreintes enregistrée non négligeable.

Une simple recherche sur Google nous donne accès à plusieurs bases de données qui contiennent plusieurs millions ou plusieurs milliards d'empreintes md5/sha1.



### 1.3. Présentation du projet

Notre projet a pour objectif de développer un **logiciel capable de générer des dictionnaires d'empreintes**, puis de rechercher un hash à l'intérieur de celui-ci.

Notre logiciel pourra fonctionner avec un ou plusieurs threads en parallèle, et sur plusieurs fonctions de hachages différentes. Cela nous permettra d'effectuer une analyse comparative en termes de performance, pour vérifier notamment si un calcul multi-threadé est plus rapide qu'un calcul mono-thread.

Nous pourrions par ailleurs analyser le fonctionnement sur des machines ayant un processeur à 1, 2, 4 ou 8 cœurs, et gérant l'hyper-threading ou non (2 cœurs virtuels par cœur physique).

Concernant l'implémentation technique, un processus principal aura la responsabilité de distribuer les calculs et d'enregistrer les résultats. Un ou plusieurs threads seront, quant à eux, chargés d'effectuer une série hachage sur des combinaisons consécutives.

Les données générées seront stockées, soit de manière brute dans un document texte, soit par l'intermédiaire d'une base NOSQL (Not Only SQL).

Les fonctions de hachages qui seront supportées sont les suivantes :

- Sum (5bits)
- Cksum (10bits)
- Md5sum (32 bits) (via le shell ou en natif)
- Sha1sum (40 bits)
- Sha224sum (224 bits)
- Sha256sum (256 bits)
- Sha384sum (384 bits)
- Sha512sum (512 bits)

Elles sont incluses par défaut dans toute distribution linux intégrant les utilitaires « GNU Coreutils ».

#### 1.4. Mis en œuvres

Parmi les concepts de programmation système Unix étudiés au cours de ce semestre, notre projet va utiliser les concepts suivants :

- **Gestion de fichiers** : Enregistrement des données générées dans un fichier.
- **Programmation multithreads** : afin de paralléliser les calculs.
- **Gestion des signaux** : arrêter proprement les calculs en cas de réception d'un signal d'arrêt, et enregistrer les derniers résultats.
- **Synchronisation** : La programmation multithread oblige à l'utilisation de mécanismes de synchronisation, notamment sur l'utilisation des variables partagées ou sur les écritures simultanées.
- **Processus** : Exécution de commandes Shell dans le programme et récupération du résultat.
- **Communication entre processus** : ~~Les calculateurs vont transmettre leurs résultats au processus principal au travers de tubes ou d'IPC~~ : Ce concept n'a finalement pas été retenu, la communication multithread étant gérée nativement.



## 2. GESTION DE PROJET

### 2.1. Entre Scrum et Extreme-Programming

Scrum<sup>3</sup> signifie mêlée au rugby. Scrum utilise les valeurs et l'esprit du rugby et les adapte aux projets de développement. Comme le *pack* lors d'un ballon porté au rugby, l'équipe chargée du développement travaille de façon collective et est soudée vers un objectif précis. Comme un demi de mêlée, le Scrum Master aiguillonne les membres de l'équipe, les repositionne dans la bonne direction et donne le tempo pour assurer la réussite du projet.



Si la vraie nature de Scrum est difficile à définir ; il est beaucoup plus simple d'expliquer la mécanique de mise en œuvre :

- Scrum sert à développer des produits, généralement en quelques mois. Les fonctionnalités souhaitées sont collectées dans le backlog de produit et classées par priorité. C'est le product Owner qui est responsable de la gestion de ce backlog.
- Une version (release) est produite par une série d'itérations appelées des sprints. Le contenu d'un sprint est défini par l'équipe, avec le Product Owner, en tenant compte des priorités et de la capacité de l'équipe. A partir de ce contenu, l'équipe identifie les tâches nécessaires et s'engage pour réaliser fonctionnalités sélectionnées pour le sprint.
- Pendant un sprint, des points de contrôle sur le déroulement des tâches sont effectuées lors des mêlées quotidiennes (*Scrum Meeting*). Cela permet au

---

<sup>3</sup> Pour une présentation plus complète de la méthodologie Scrum, un article y est consacré à l'adresse suivante : <http://www.studio-dev.fr/blog/gestion-de-projet/gestion-de-projet-avec-scrum.htm>



ScrumMaster, l'animateur chargé de faire appliquer Scrum, de déterminer l'avancement par rapport aux engagements et d'appliquer, avec l'équipe, des ajustements pour assurer le succès du sprint.

- A la fin de chaque sprint, l'équipe obtient un produit partiel (un incrément) qui fonctionne. Cet incrément du produit est potentiellement livrable et son évaluation permet d'ajuster le backlog pour le sprint suivant.

Scrum est une méthode de gestion de projet très complète qui définit entre autre des rôles, des artefacts et des cérémoniaux. Nous n'aurons pas la possibilité d'intégrer tous ces éléments durant la réalisation de ce projet, pour des raisons de manques de temps et de moyens.

Cependant, nous allons tenter de nous en rapprocher le plus possible, notamment concernant les valeurs et les **bonnes pratiques à adopter** (issues pour la plupart du manifeste agile). Nous suivrons aussi à la lettre la méthode de gestion des priorités au niveau des tâches (**product backlog**), et nous participerons en commun à l'élaboration du **planning** et des **sprints**.

## 2.2. Estimation et planification des tâches

La **product backlog** est un document référençant l'ensemble des tâches à effectuer pour arriver à la version finale du produit.

Chaque tâche est caractérisée par au moins 3 éléments :

- Son état (à faire, en cours, à tester, fini)
- Sa priorité
- Son estimation (en termes de points, comparativement aux autres)

Ces éléments ont été définis en communs par l'équipe. Le backlog ne définit pas qui doit réaliser telle ou telle tâche, **chaque développeur inoccupé doit simplement s'affecter la tâche ayant la priorité la plus élevée restant à faire** (des exceptions étant exceptionnellement tolérées).

L'estimation permet d'avoir un ordre d'idée de la complexité de la tâche. Dans une gestion de projet complexe, un ScrumMaster connaissant la vélocité de l'équipe pourrait transcrire ces résultats en heures, mais ce n'est que très peu précis sur un premier projet. Nous avons donc décidé d'omettre cette étape.

Le planning découle alors directement d'un découpage en fonction des priorités et des estimations. Notre objectif est de travailler sur des **sprints d'1 semaine**, donc les objectifs sont les suivants :

- Sprint 0 : Choix du sujet, mise en place de l'équipe et analyse fonctionnelle
- Sprint 1 : Préparation environnement de développement et recherches technologique
- Sprint 2 : Développement des algorithmes et des fonctions du cœur du logiciel
- Sprint 3 : Développement des fonctions annexes, benchmarks, et finalisation du rapport.

Un point important à préciser est qu'un backlog est **ouvert au changement**. Il est extrêmement déconseillé de modifier les tâches prévues dans le sprint en cours, en revanche les tâches des sprints suivants peuvent être modifiées sans conséquences.

Voici donc un aperçu du backlog de produit, à la fin des développements :

	User Story	Priorité	Estimation
<b>Sprint 0</b>	Mise en place de l'équipe	1	-
Semaine 0	Rédaction de la proposition de sujet	2	-
	Analyse : Définition des besoins	30	-
	Analyse : Architecture UML (diag classes)	31	-
	Rédaction du rapport : Partie 1 - Analyse	50	-
	Rédaction du rapport : Partie 1 - Gestion projet	51	-
	Rédaction du rapport : Partie 1 : Conception	52	-
<b>Sprint 1</b>	Mise en place de l'environnement de développement (VM)	80	3
Semaine 1	Intégration SVN	81	2
	Recherche : Tests unitaires C++	100	8
	Architecture : Structure générale du logiciel	100	13
	Recherche : Utilisation d'une base de donnée Nosql	101	13
	Recherche : Enregistrement optimisé dans un fichier	102	5
	Recherche : Méthode la plus rapide pour calculer une empreinte	103	5
	Développement : Mise en place de l'architecture multi-thread (multi-proc?)	105	8
<b>Sprint 2</b>	Algorithme : Prédiction brute force (quelle chaîne arrive après N itérations à partir de la chaîne X)	200	8
Semaine 2	Algorithme : Quelle est la chaîne suivant la chaîne X	201	5
	Préparer template c++ Singleton et Observer/Observable	202	5
	Recherche : Méthode optimale pour transférer les données entre processus (ou threads)	300	5
	Développement : Gestion des signaux	301	5
	Développement: Gestion des threads/pipes	302	13
	Intégration des différents éléments, mise en place Synchronisation	303	8
<b>Sprint 3</b>	Interface : Configuration du logiciel (console)	400	5
	Interface : Affichage pendant les calculs	400	5
Semaine 3	Développement : Calcul des performances	401	3
	Développement : Script bash pour retourner les résultats	402	3
	Finalisation : Benchmarks	700	2*2
	Rédaction du rapport : Partie 2 - Documentation utilisateur	900	5
	Rédaction du rapport : Partie 2 - Assurance qualité	901	3
	Rédaction du rapport : Partie 2 - Améliorations possibles	902	2
	Préparer démonstration	903	3

## 3. CONCEPTION PRÉLIMINAIRE

### 3.1. Définition des cas d'utilisations

Voici les différents cas d'utilisations auxquels devra répondre notre logiciel :

- L'utilisateur génère un dictionnaire d'empreinte
  - L'utilisateur choisit le type de hachage à utiliser
  - L'utilisateur choisit le nombre de threads à utiliser en simultanées
  - L'utilisateur choisit le format d'enregistrement des données
  - L'utilisateur indique s'il souhaite continuer ou reprendre à zéro la génération
- L'utilisateur visualise les vitesses de calculs
- L'utilisateur recherche une empreinte dans la base de données

### 3.2. Architecture logicielle

Voici une description simplifiée du fonctionnement du programme

La méthode *Main* commence par instancier 2 objets transverses :

- Le gestionnaire de *signaux* qui va intercepter les signaux SIGINT <CTRL+C>
- Le gestionnaire de *logs*, qui permet en fonction de sa configuration d'afficher les logs à l'écran ou de les enregistrer dans un fichier. Il gère par ailleurs plusieurs niveaux d'erreurs.

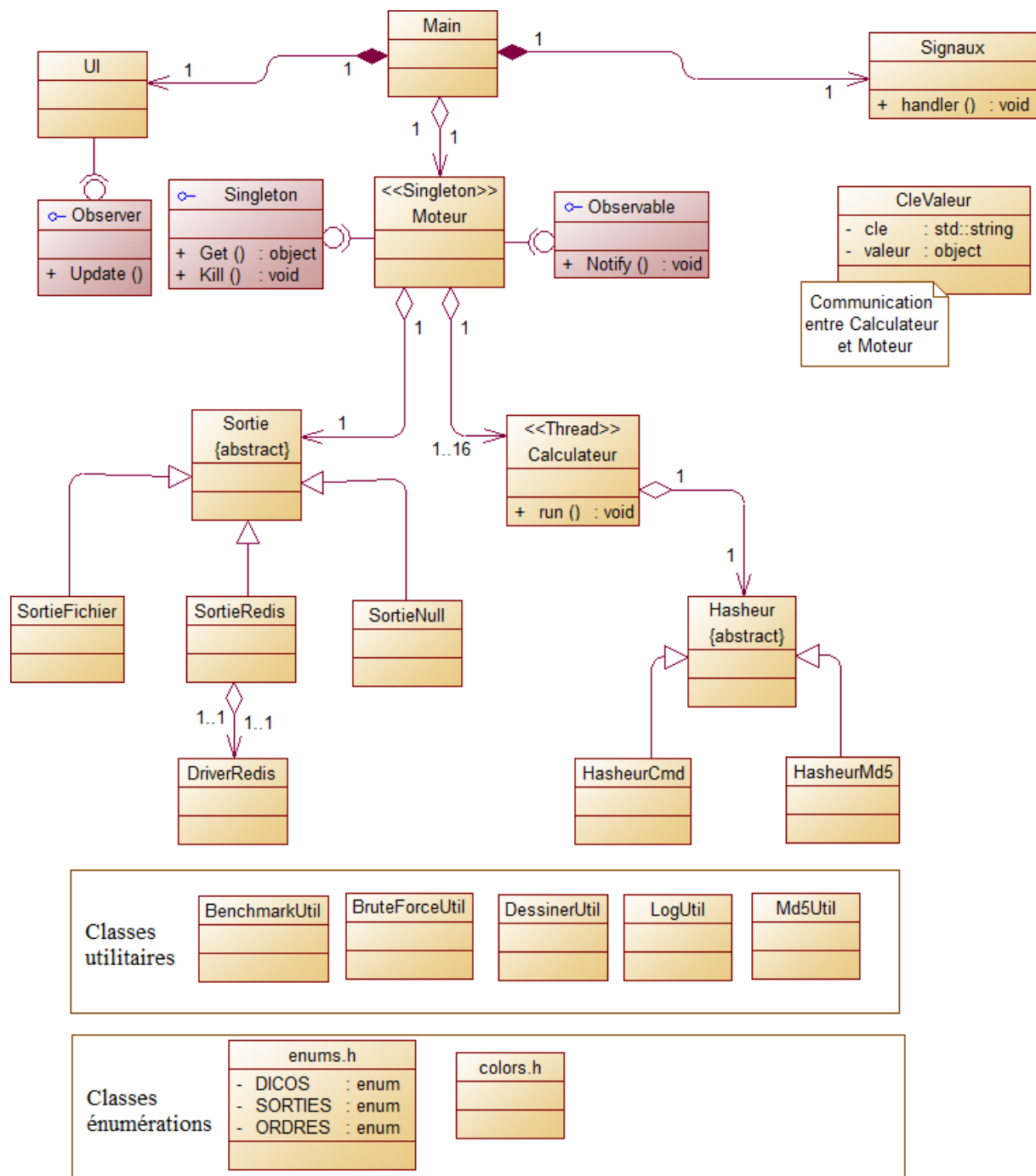
Un objet de la classe *UI* est ensuite généré. Il sera chargé de gérer l'affichage de la console, et il sera à l'écoute des événements du *Moteur* (pattern *Observer/Observable*)

Le *moteur*, classe principale de l'application, que l'on pourrait comparer au chef d'orchestre, est instancié par un *Singleton* pour faciliter son accès aux différentes classes (Cela introduit un couplage fort, mais cette classes étant indispensable, ce n'est pas dommageable.)

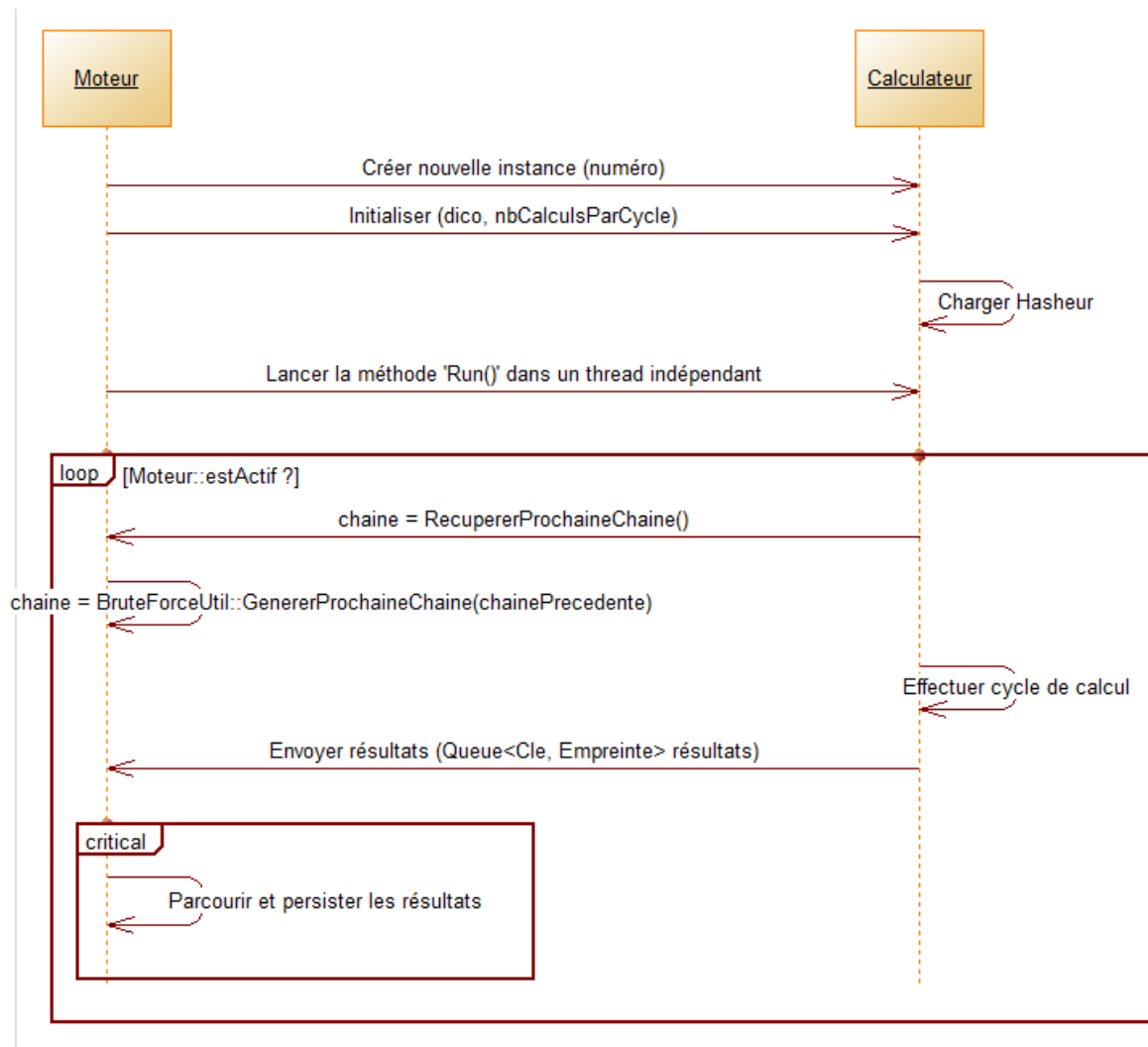
Ce dernier est chargé d'initialiser et d'ouvrir la *sortie* choisie (écriture dans un fichier, dans une base de donnée REDIS, ou nulle part), puis de lancer les N threads qui vont exécuter en boucle la méthode *run()* d'instances de *Calculateur*.

Lors de l'initialisation d'un calculateur, celui-ci va instancier un objet *Hasheur* du type demandé, et il l'utilisera pour effectuer chacun de ces calculs d'empreintes.

Voici le diagramme UML de notre architecture :



Voici un diagramme de séquence montrant la communication entre le moteur et l'un de ses calculateurs :












### 3.3. Environnement de travail

Un environnement sur **machine virtuelle VmWare** (Debian) a été préparé afin que l'ensemble des membres de l'équipe puisse travailler avec exactement les mêmes outils.

L'IDE **Netbeans** permettra de développer puis de compiler nos programmes écrits en C++. Il génère automatiquement des fichiers **makefile** (en gérant plusieurs profils) intégrant les librairies et les liens nécessaires.

Les sources du projet sont intégrées au sein d'un dépôt **SVN** pour faciliter le travail multi-développeur et éviter au maximum les conflits. Un fournisseur de dépôt gratuit a été sélectionné, **Assembla** ; un site qui offre par ailleurs plusieurs outils tel qu'un Wiki, gestionnaire de Tickets, etc ...

<b>epsiprojetunix2011</b> is an open source project powered by Assembla Assembla offers secure, commercial-quality <a href="#">Hosted SVN</a> and <a href="#">GIT Repositories</a> , <a href="#">Ticketing &amp; Bug Tracking</a> , and <a href="#">Wikis</a> . Host your community projects for free or try our subscription plans free for 30 days.					
<a href="#">Get a Workspace</a>					
<b>Changesets</b>					
Apr 17	10:00 UTC		<a href="#">studio-dev</a>	committed [47] Recherche et correction fuites mémoires Suppression logs inutiles	<a href="#">View</a> »
	22:53 UTC		<a href="#">studio-dev</a>	committed [46] Correction SVN	<a href="#">View</a> »
	22:51 UTC		<a href="#">studio-dev</a>	committed [45] Ajout des scripts Correction de bugs	<a href="#">View</a> »
Apr 16	18:23 UTC		<a href="#">studio-dev</a>	committed [44] Création Tag Beta1	<a href="#">View</a> »
	18:19 UTC		<a href="#">studio-dev</a>	committed [43] Refactoring et nettoyage du code	<a href="#">View</a> »
	17:58 UTC		<a href="#">studio-dev</a>	committed [42] Version Beta	<a href="#">View</a> »
	13:11 UTC		<a href="#">studio-dev</a>	committed [41] UI "Mode calcul" codé. Bug en environnement multi-thread => Il faut trouver u...	<a href="#">View</a> »
	00:33 UTC		<a href="#">studio-dev</a>	committed [40] Design Pattern Observer/Observable entre l'UI et le MOTEUR (à terminer)	<a href="#">View</a> »
	23:55 UTC		<a href="#">studio-dev</a>	committed [39] Travail sur l'UI de configuration. Ajout de la fonctionnalité "Rechercher der...	<a href="#">View</a> »

Un de nos objectifs facultatif est d'interfacer notre logiciel avec une base de données NOSQL (**Not Only SQL**). Ce type de base de données serait parfaitement adapté à notre projet. Elles permettent en effet d'effectuer des opérations d'écriture très rapidement, en minimisant les fonctionnalités de relations, transactions, vérifications, etc... De surcroit, utilisées sur un système de production, elles offrent des possibilités de distribution de données très performantes.

Notre choix initial se base sur **Redis**, une base orientée « clé/valeur » implémentant des types avancés tels que les listes. Ce mode d'enregistrement nous semble être en parfaite adéquation avec notre besoin. La question qui reste en suspens est de savoir si la vitesse d'enregistrement sera suffisante pour ne pas générer un goulot d'étranglement.

Chaque fonction algorithmique sera testée unitairement en utilisant le principe du TDD (**Test Driven Development**) : le test doit être écrit avant le développement de la fonction, et celle-ci est corrigée/améliorée tant que tous les tests ne sont pas concluants.

La librairie de test unitaire choisie est **CppTest**, qui correspond parfaitement à notre besoin sans être trop complexe d'utilisation.

Enfin, l'utilitaire **Doxygen** a été utilisé pour générer la documentation technique du code.

## 4. DOCUMENTATION UTILISATEUR

### 4.1. Prérequis

Le programme a été développé sur un environnement de type **Debian** (Ubuntu 10.10).

Le programme ainsi que les librairies externes utilisées, n'ont été compilés et testés que sur cet environnement.

Il est par ailleurs conseillé de lancer le programme dans un terminal classique (gnome-terminal, konsole), avec la taille par défaut (80\*25). Si ces conditions ne sont pas remplies, l'affichage pourrait s'en trouver détérioré.

Pour utiliser la fonctionnalité d'enregistrement en base NOSQL, il est nécessaire d'avoir préalablement installé un **serveur REDIS**.

Pour cela, il suffit de le télécharger depuis les dépôts :

```
apt-get install redis-serv
```

Par ailleurs, il est nécessaire de faire attention à l'espace disque disponible, la génération de dictionnaire pouvant créer des fichiers de plusieurs GO en quelques minutes.

### 4.2. Génération des dictionnaires

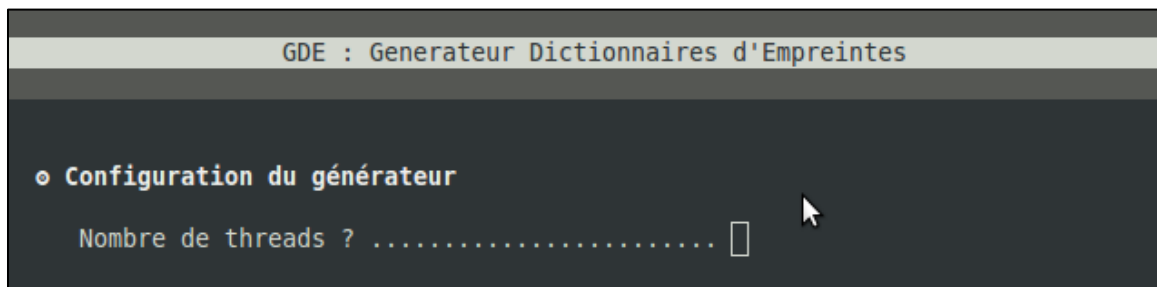
Une fois le programme compilé via l'environnement de compilation Netbeans, il suffit de lancer l'exécutable :

```
./projetunixhashgenerator
```

Dans un premier temps, le programme va demander à l'utilisateur quelques informations pour configurer la génération du dictionnaire. Ensuite, le programme calculera de manière ininterrompue des empreintes.

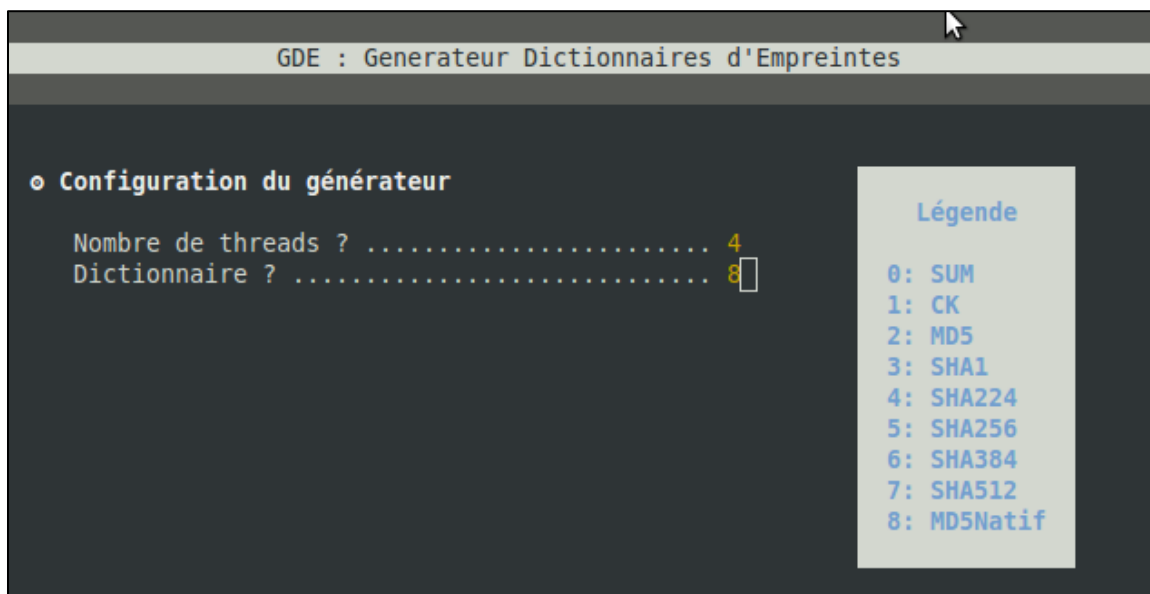


### Etape 1 : Choix du nombre de thread



L'utilisateur doit entrer le nombre de calculateurs qui seront créés en tant que threads indépendant.

### Etape 2 : Choix du dictionnaire

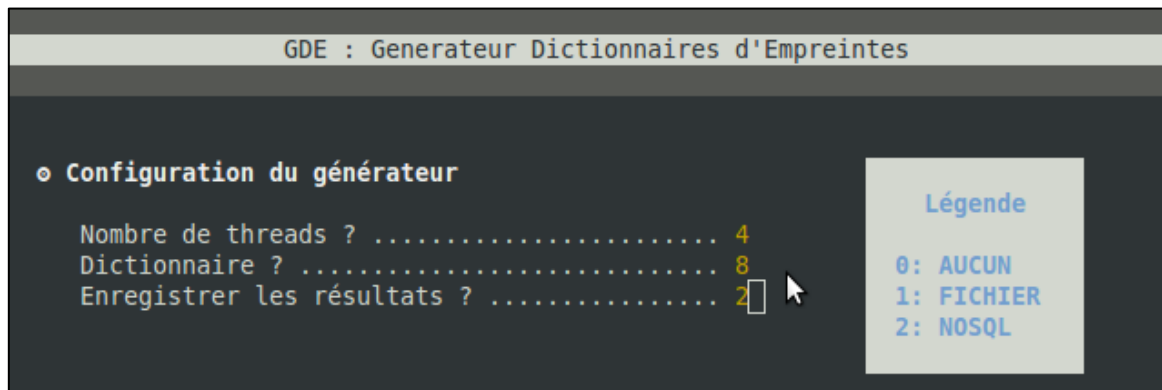


L'utilisateur doit maintenant sélectionner le dictionnaire qu'il souhaite générer.

Les différents dictionnaires (=algorithmes) sont à classer en 2 catégories :

- Pour les dictionnaires 0 à 7, la génération de l'empreinte va être déléguée à un exécutable inclus dans le projet GNU coreutils. L'avantage est qu'il n'est pas nécessaire de coder l'algorithme de hachage, mais l'inconvénient est que tous ses appels à des fonctions du Shell ralentissent énormément le programme.
- Le 8<sup>ème</sup> dictionnaire, nommé MD5Natif, offre une méthode de hachage directement codée et compilée en C dans le programme. La vitesse de calcul est donc incontestablement plus rapide (jusqu'à 2 000 fois supérieure).

### Etape 3 : Choix de la sortie pour l'enregistrement



Cette étape permet à l'utilisateur de choisir s'il souhaite enregistrer les résultats des calculs, et si oui sur quel support.

- **Aucun** : les résultats ne sont pas enregistrés (ce qui idéal pour effectuer des comparaisons de vitesse de calcul, car l'on est certain que la vitesse de génération ne sera pas bridée par la vitesse d'écriture).
- **Fichier** : Les résultats sont enregistrés dans un fichier dictionnaire nommé '#.hash' (où # correspond au numéro du dictionnaire). La vitesse d'enregistrement est assez rapide, de l'ordre de 500 000 lignes par seconde, c'est-à-dire environ **1 giga/mn**. Il est possible de regarder ou de suivre le remplissage du fichier avec un simple `tail -f`.

```
dwmal a17ca9157fe5fb1795df8866cf47c676  
dwmam b7e9ae6f131a0a10c9d6f6d7ad6fd978  
dwman db25c5cbf6a24d79214172ca5891e1ea  
dwmao 85176e990af2b1bddde0434abf152252  
dwmap 430307a310ac507f0581c665c2b6b99c  
dwmaq 605263e5f988eebdd874ef70d6067ae4  
dwmar 059cf4392a5a80e5790e93386fe003f5
```

- **NOSQL** : Les résultats sont ici persistés dans la base de données NOSQL Redis. Les vitesses d'enregistrements sont moins rapide qu'en écrivant dans un fichier, en revanche l'optimisation est portée sur la **fonction de recherche**. Alors qu'un `grep` sur un fichier de plusieurs giga-octets peut prendre plusieurs secondes, une requête sur la base de données donne un résultat quasi-immédiatement. De plus, le format de stockage physique est optimisé, et offre des fonctions de clustering (indispensable si l'on veut générer un dictionnaire conséquent, de plusieurs go de données). Il est possible de suivre les enregistrements en lançant le client `redis-cli`, puis en tapant la commande `MONITOR`

```
+1302885280.882579 (db 8) "SADD" "9d032343a8e7b470303a8ecbc7f04457" "bmenk"  
+1302885280.882674 (db 8) "SADD" "4a4b691d40020e9fcf7c5393f904b37b" "bmenl"  
+1302885280.882761 (db 8) "SADD" "9b127809af5d3e26e17fb5c76371752a" "bmenm"  
+1302885280.882949 (db 8) "SADD" "16f43adc9dea4d2a09d8d41bbd99c41a" "bmenn"  
+1302885280.883043 (db 8) "SADD" "74088ab30a34725e69ed5a2ee9f629e7" "bmeno"  
+1302885280.883127 (db 8) "SADD" "a4d9bdc6a8f4f28ed047b8c013127b4" "bmenn"
```

#### Etape 4 : Reprendre ou non la génération



Cette dernière étape permet de choisir si l'on veut recommencer la génération des empreintes à zéro, ou si l'on veut reprendre là où l'on s'était arrêté.

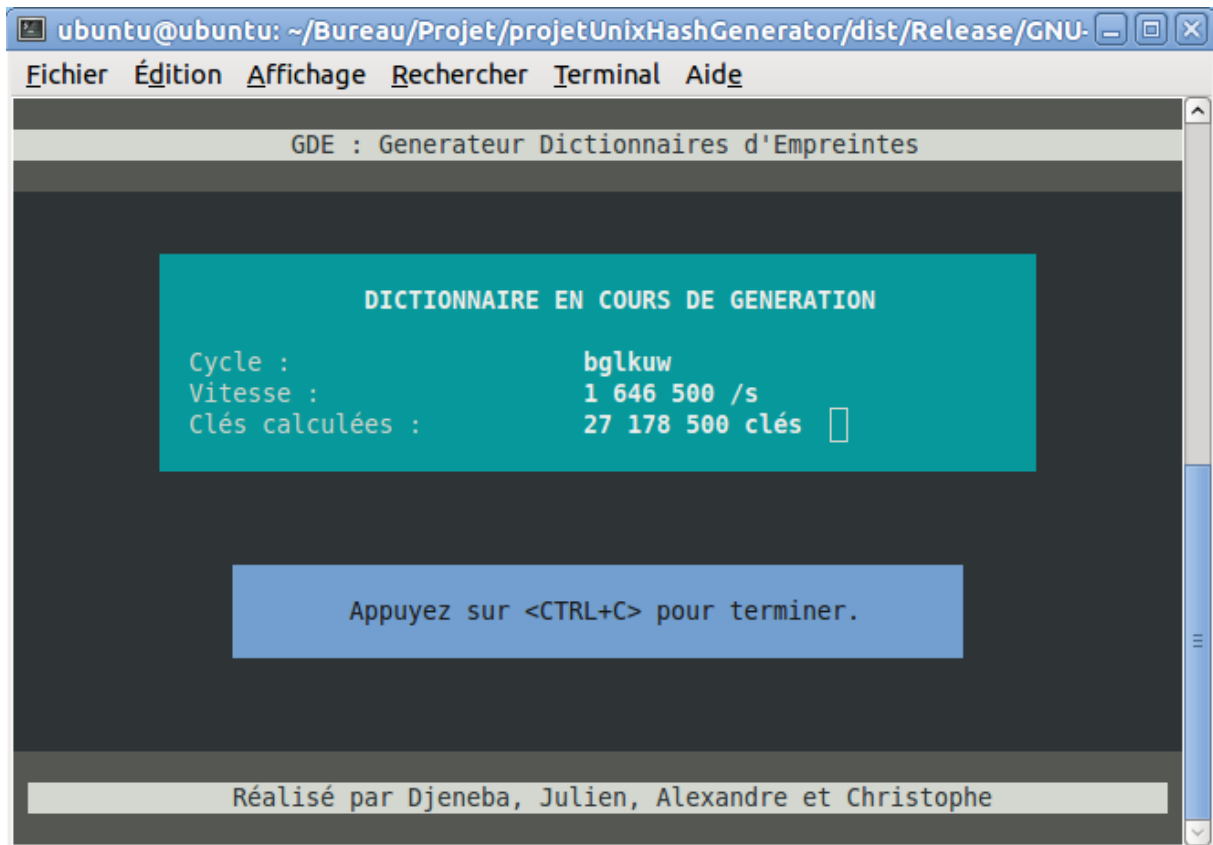
- Dans le premier cas, le dictionnaire sera vidé avant de recommencer les calculs.
- Dans le second cas, le moteur va rechercher quelle a été la dernière clé traitée, puis partir de la suivante. Cette fonction permet de pouvoir arrêter le générateur quand on le souhaite et de pouvoir reprendre la génération ultérieurement.

Une fois validé, le moteur va s'initialiser puis lancer les calculateurs.

La génération des dictionnaires commence.

## GÉNÉRATION DES DICTIONNAIRE EN COURS

Une fois le moteur initialisé avec succès, un nouvel écran apparaît permettant de suivre l'avancement de la génération du dictionnaire.



L'écran récapitule 3 informations mises à jour toutes les secondes :

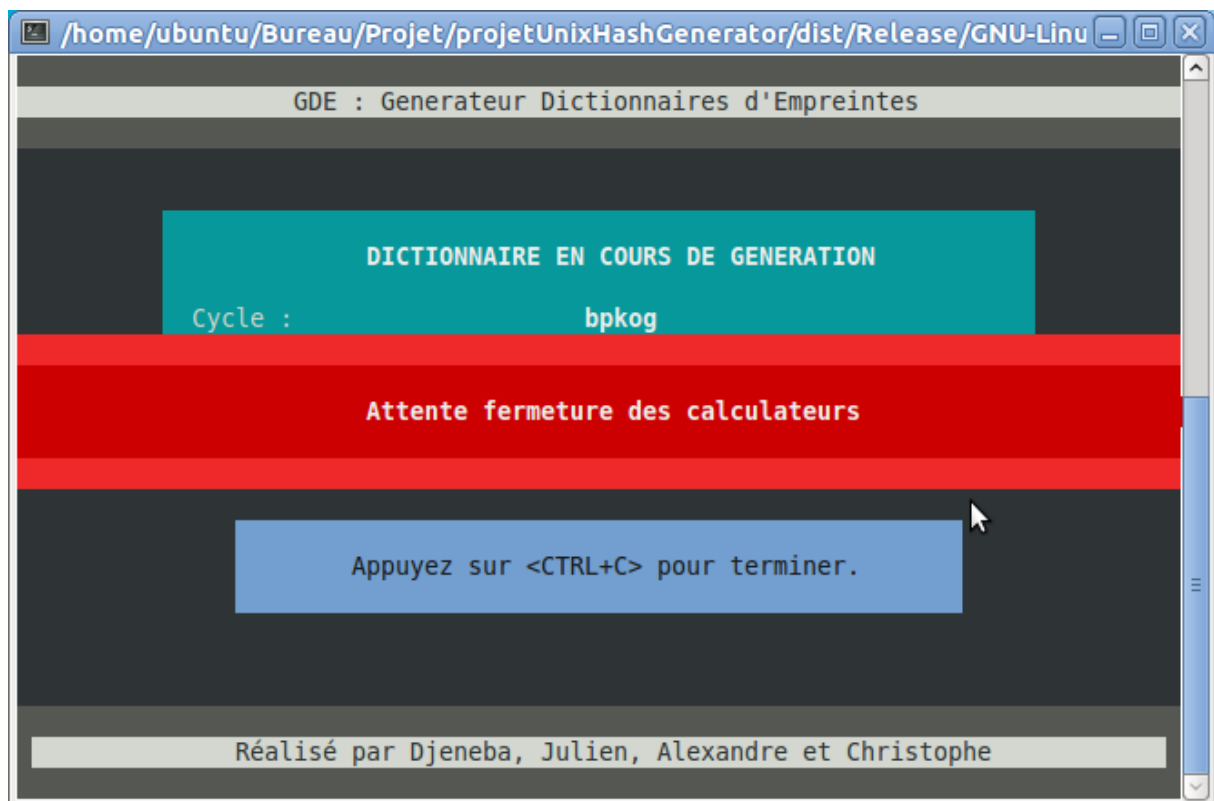
- **Cycle** : La dernière chaîne de caractères transmise à un calculateur, à partir duquel il va effectuer son cycle de calcul (suivant la configuration, il va effectuer par exemple 200 générations à partir de cette chaîne avant d'envoyer le résultat au moteur).
- **Vitesse** : La vitesse instantanée de calcul
- **Clés calculées** : Le total des clés calculées durant cette session

Le programme va tourner en boucle tant qu'aucun signal n'est reçu.

En conséquence, pour demander au programme de se terminer, il faut lui envoyer un signal, le plus simple étant *SIGINT* (combinaison *CTRL+C*).

Le moteur ne va pas couper les threads directement, il va attendre que les calculateurs aient fini leur cycle en cours puis sauvegarder les résultats et fermer connexions/fichiers ouverts.

Ce comportement garantit que les dictionnaires ne soient pas altérés à cause d'une interruption brutale.



---

#### QUE FAIRE EN CAS D'ERREUR ?

Un fichier de log est disponible dans le même répertoire de l'exécutable (nommé *gdeYYYYMMAA.log*). Il permettra de voir quelles sont les dernières opérations qui se sont exécutées, et de savoir si une exception a été interceptée.

Le niveau d'enregistrement est placé par défaut sur « *INFO* », il est possible de placer ce niveau à « *TRACE* » pour obtenir plus d'informations (mais cela ralentit l'exécution du programme).

### 4.3. Recherche de résultats

La recherche des résultats dans les dictionnaires ne faisait pas parti du cahier des charges initial.

Nous avons cependant mis en place deux scripts **Bash** offrant cette fonctionnalité, ce qui permet de vérifier que le fonctionnement du programme est correct.

#### Installation :

- Il faut dans un premier temps rendre les deux scripts exécutables.  
Ceux-ci sont situés dans le dossier /scripts/  
Pour cela il faut leur appliquer la commande :

```
chmod +x /...../projetUnixHashGenerator/scripts/*
```

- Par souci de commodité, il est conseillé de rajouter les scripts dans le PATH, pour que le Shell puisse les trouver directement.

```
export PATH=$PATH:/...../projetUnixHashGenerator/scripts
```

- Les deux scripts sont maintenant prêts à être exécutés :
  - **find\_fichier** : Permet de rechercher dans la sortie Fichier (attention : avant de lancer l'utilitaire, il faut se placer dans le répertoire contenant les dictionnaires)
  - **find\_redis** : Permet de rechercher dans la base de donnée Redis

#### Utilisation :

Une fois l'utilitaire lancé, il suffit de suivre les instructions à l'écran :

1. Choisir l'opération à effectuer (uniquement pour la recherche dans les fichiers)
2. Sélectionner le dictionnaire sur lequel travailler
3. Saisir ce que l'on souhaite rechercher

## Exemples :

Rechercher dans le dictionnaire « MD5 Natif » la **clé** correspondant à une empreinte donnée :

```
ubuntu@ubuntu: ~/Bureau/Projet/projetUnixHashGenerator/dist/Release/GNU-
Fichier Édition Affichage Rechercher Terminal Aide
#####
##### Recherche dans un dictionnaire Fichier #####
#####

Quelle opération souhaitez-vous effectuer
  1: Rechercher une empreinte, à partir de sa clé
  2: Rechercher une clé, à partir de son empreinte
> 2

Sur quel dictionnaire voulez-vous procéder la recherche ?
0: SUM
1: CK
2: MD5
3: SHA1
4: SHA224
5: SHA256
6: SHA384
7: SHA512
8: MD5 Natif
> 8

Saisissez l'empreinte : 534b44a19bf18d20b71ecc4eb77c572f

=> Clé liée à l'empreinte 534b44a19bf18d20b71ecc4eb77c572f = alex
```

Rechercher dans le dictionnaire « MD5 Natif » **l'empreinte** correspondant à la clé donnée :

```
ubuntu@ubuntu: ~/Bureau/Projet/projetUnixHashGenerator/dist/Release/GNU-
Fichier Édition Affichage Rechercher Terminal Aide
#####
##### Recherche dans un dictionnaire Fichier #####
#####

Quelle opération souhaitez-vous effectuer
  1: Rechercher une empreinte, à partir de sa clé
  2: Rechercher une clé, à partir de son empreinte
> 1

Sur quel dictionnaire voulez-vous procéder la recherche ?
0: SUM
1: CK
2: MD5
3: SHA1
4: SHA224
5: SHA256
6: SHA384
7: SHA512
8: MD5 Natif
> 8

Saisissez la clé : djene

=> Empreinte de djene = 2d76330483a91ab5f2823454a1f12dd5
```



## 5. ASSURANCE QUALITÉ

### 5.1. Conventions de codage

Voici la convention de codage mis en place sur notre projet.

Plusieurs points sont tirés du « **Google C++ Style Guide** <sup>4</sup> », la convention de codage mis en place sur les projets réalisés par Google.

---

#### CONVENTION DE NOMMAGE

- Les noms des fonctions, variables, fichiers, classes, etc. doivent être descriptifs et succincts.
- Les types commencent par une majuscule ; une lettre capitale est utilisée pour chaque nouveau mot : `MaMagnifiqueClasse`, `MonSublimeEnum` (notation CamelCase)
- Les variables sont écrites en minuscule, avec un `_` entre chaque mot : `ma_variable_locale`.
- Les variables de classes doivent être préfixées d'un `_` : `_ma_variable_de_classe`
- Les constantes sont écrites en majuscule : `MACONSTANTE`
- Les nom des fonctions ont la même règle que les classes : `MaSuperbeFonction()`

---

#### CONVENTION DE CODAGE

##### Entêtes

- Utiliser l'extension `.h` pour les fichiers d'entête
- Chaque fichier d'entête doit être protégé aux multiples inclusions grâce à un `#define`. Le format à utiliser est : `#define MACLASSE_H`
- N'utiliser les méthodes en ligne que si le cœur de la fonction est inférieur à 10 lignes
- Les paramètres d'une fonction doivent être dans l'ordre suivants : entrées .... sorties

##### Classes

- Utiliser une fonction `init()` pour initialiser des variables dans un constructeur
- Il faut définir un constructeur par défaut si la classe contient des variables membres (sinon le compilateur le fait pour vous, badly ).
- Préciser obligatoirement le type des variables membres (`private`, `protected`, `public`...)

##### Autres mécanismes

- Privilégier le passage de paramètres avec le mot clé `const`
- Utiliser les classes amies avec parcimonie

---

<sup>4</sup> <http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml>

## Commentaires

- Toujours documenter les méthodes via la syntaxe `/** */` (Doxygen)
- A l'intérieur d'une fonction : **"Never comment code, always comment decisions"**

## 5.2. Librairie de tests unitaires CppTest

CppTest est une librairie permettant de faire du test unitaire, c'est-à-dire de faire un test dit « boîte noire » sur une fonction particulière du programme. Nous avons choisi cette librairie car elle fait partie des librairies qui sont simples d'utilisation, tout en restant très performantes, et répondant parfaitement à notre besoin.

En effet, CppUnit permet, grâce à la macro « `TEST_ASSERT()` », de pouvoir tester de façon efficace nos méthodes et les résultats qu'elles sont censées renvoyer. Elle offre aussi plusieurs formats de sortie comme le texte simple ou l'HTML pour générer des rapports.

Le mode *verbeux* permet d'afficher un rapport de nos tests très clair, avec le pourcentage de réussite et les tests qui sont en échec.

```
$ ./projetunixhashgenerator test
TestBruteForce: 2/2, 100% correct in 0.000037 seconds
TestHasheurMd5: 1/1, 100% correct in 0.000016 seconds
TestHasheurCmd: 3/3, 100% correct in 0.074070 seconds
TestDriverRedis: 3/3, 66% correct in 0.002096 seconds
    Test:      testConnexion
    Suite:     TestDriverRedis
    File:      TestDriverRedis.cpp
    Line:      40
    Message:   r.ping()==true
Total: 9 tests, 88% correct in 0.076219 seconds
```

Au final, nous avons utilisé cette librairie pour tester les fonctions principales de notre programme, telles que les algorithmes (brute-force), les méthodes de hashages, ou encore le driver de connexion à la base de donnée Redis.

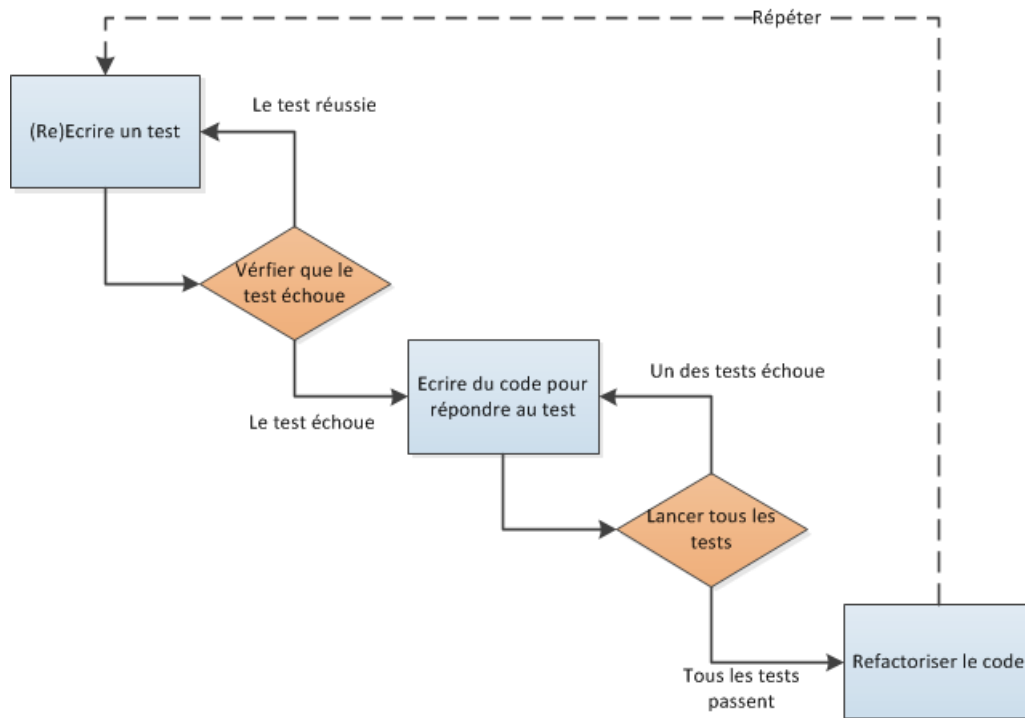
*Les tests sont visibles dans le dossier virtuel 'cpptestclass' de netbeans, où en regardant les fichiers TestXxxx.cpp*

## 5.3. Méthodologie Test Driven Development

Le Test Driven Development (ou **développement piloté par les tests**) est une méthode de développement de logiciel qui préconise l'écriture des tests unitaires avant d'écrire le code source de la fonction.

Cette méthode est conseillée par les méthodes agiles, et nous avons décidé de la suivre sur certaines fonctions développées dans ce projet.

Tout d'abord, voici une schématisation du cycle de développement en utilisant la méthodologie TDD :



Nous avons notamment mis en pratique cette méthode pour les fonctions touchant aux algorithmes de calcul Brute force qui sont parfaitement adaptés (*BruteForceUtil* :: *calculerNextChaine()* par exemple) :

- Ils sont très faciles à tester (une chaine en entrée, une chaine en sortie)
- Le développement de l'algorithme étape par étape est naturel
  - Vérifier dans un premier temps que la chaine suivant 'a' est bien 'b'
  - Vérifier ensuite que la chaine suivant 'aa' est 'ab'
  - Vérifier ensuite que la chaine suivant 'az' est 'ba'
  - Vérifier ensuite que la chaine suivant 'z' est 'aa'
  - Vérifier ensuite que la chaine suivant 'zz' est 'aaa'
  - => Chacune de ses étapes est un cas particulier qui demande un code supplémentaire.
- Améliorer un algorithme entraine un gros risque de régression, totalement couvert par cette méthode de développement.

## 6. RÉSULTATS

### 6.1. IMPACTS DE LA PROGRAMMATION MULTITÂCHE

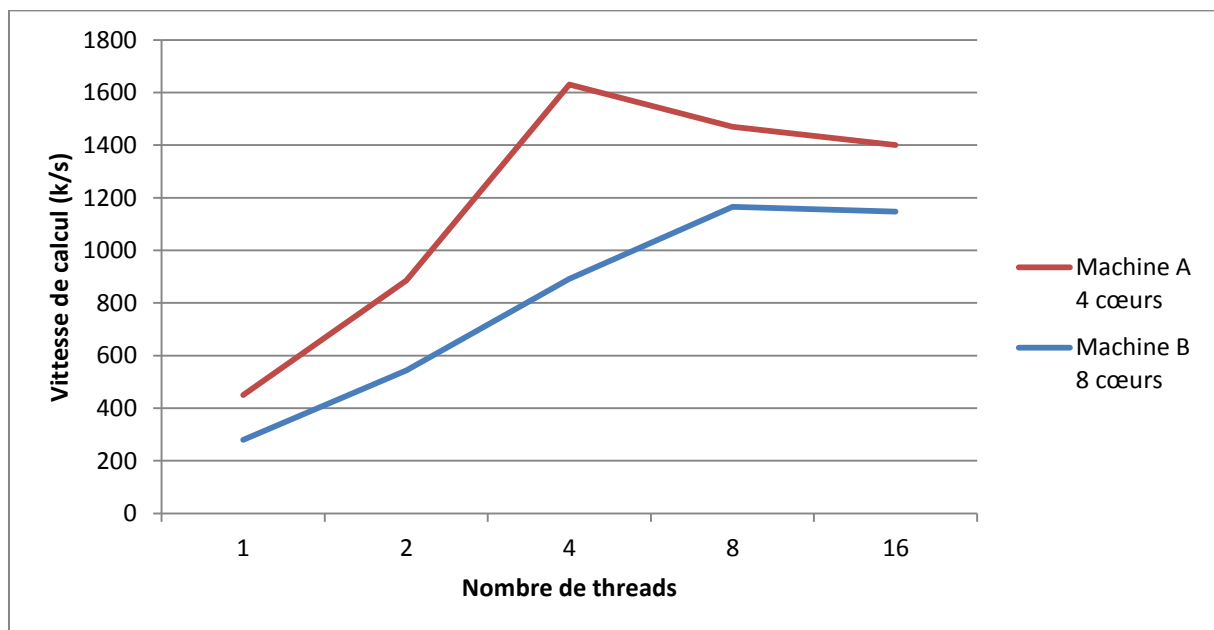
Une fois le logiciel mature, nous avons effectué une série de tests sur 2 machines différentes, afin d'évaluer les bénéfices de la programmation multitâche.

Voici le protocole de test ainsi que les machines utilisées :

Machine A	Machine B
Processeur : Intel i5 2500k : <b>4 cœurs</b> à 3.9ghz	Processeur : i7 mobile : 4 cœurs à 2.2ghz avec HyperThreading (= <b>8 cœurs virtuels</b> )
Disque dur : SSD	Disque dur : SSD
Ram : 8go	Ram : 8go
<b>Fonctionne à travers une machine virtuelle VMWare sur Ubuntu 10.1 (VT-X activé)</b>	OS : MacOSX

Pour évaluer les vitesses de calcul, nous avons effectué plusieurs mesures en configurant le logiciel avec 1, 2, 4, 8 puis 16 threads, en choisissant à chaque fois de ne pas enregistrer les résultats, et d'utiliser l'algorithme MD5Natif (le plus rapide).

Voici les résultats obtenus :



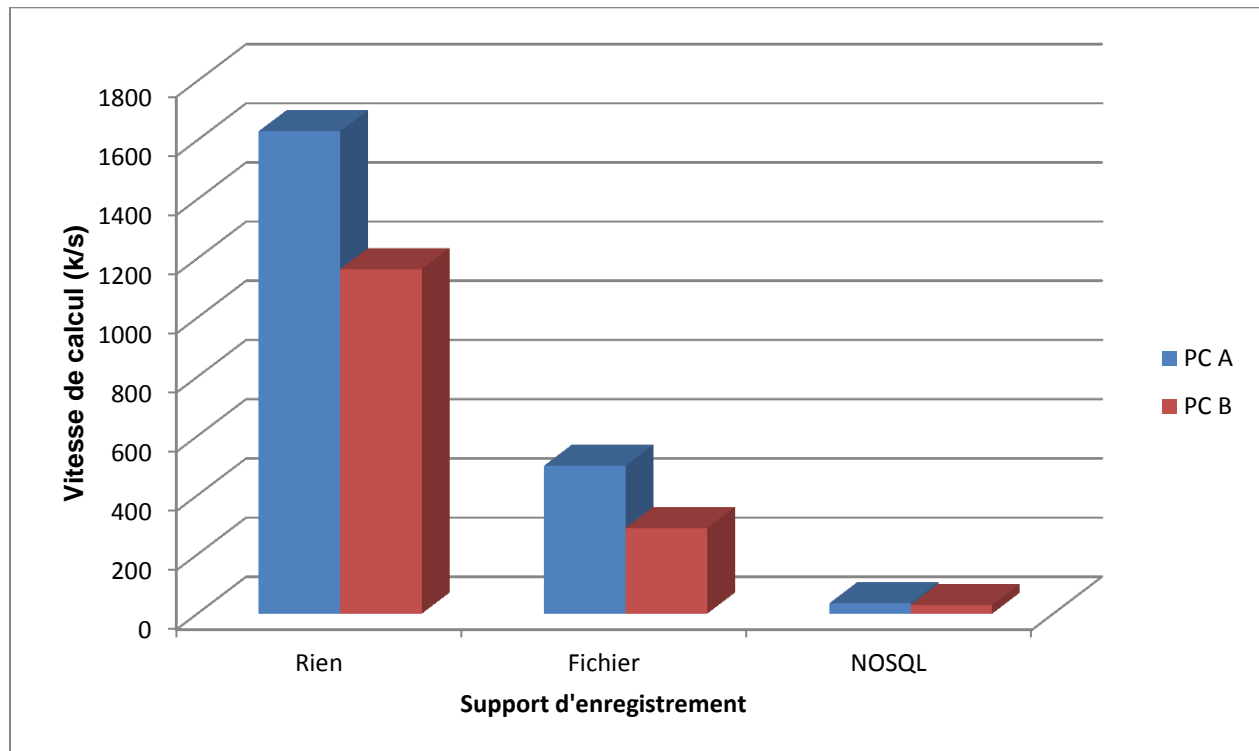
Interprétation :

- La vitesse de calcul est optimale lorsque le **nombre de threads utilisés** correspond à **l'architecture de la machine**.
- En rajoutant des threads, la vitesse de calcul décline.
- L'utilisation du multithreading permet des calculs environ **4x** plus rapide.

Dans un second temps, nous avons voulu comparer les différents supports d'enregistrement disponibles.

Le protocole de test consiste cette fois-ci à configurer le logiciel avec le nombre de threads optimum (4 pour A, 8 pour B), de choisir successivement les 3 types de sorties, toujours avec le dictionnaire MD5Natif. Plusieurs séries de mesures seront prises pour obtenir une moyenne.

#### Résultats obtenus :



#### Interprétation :

- De manière prévisible, l'enregistrement des résultats crée un **goulot d'étranglement** qui ralentit considérablement la vitesse de calcul.
- On remarque que les vitesses avec enregistrement fichier sont très proches des vitesses de calcul avec 1 seul thread. Cela peut s'expliquer car l'enregistrement se fait dans une **zone d'exclusion mutuelle** pour éviter les accès concurrents aux fichiers.
- La vitesse d'enregistrement dans la base de données Redis plafonne à 35 000 /s. Nous n'avons pas vérifié si une configuration plus précise de la base de données pouvait augmenter ce rendement.

## 6.2. Améliorations possibles

En effectuant la série de benchmark, nous avons remarqué que l'enregistrement des résultats dans un fichier réduit énormément les résultats en raison de la zone d'exclusion mutuelle mise en place. Il serait alors intéressant de réduire ce phénomène en demandant à chaque calculateur (c'est-à-dire à chaque thread) d'enregistrer lui-même les résultats de ses calculs dans un fichier indépendant.

On garderait ainsi une intégrité des fichiers, tout en optimisant encore plus les vitesses.

Un autre point intéressant serait d'utiliser une librairie dédiée aux interfaces graphiques en mode console (**NCurse** par exemple), ce qui permettrait d'avoir un affichage similaire, mais avec une plus grande compatibilité (A noter que l'interface ressort actuellement parfaitement sur un terminal MacOSX et Linux type Debian).

Enfin, de nombreuses **optimisations** peuvent être réalisées pour rendre les calculs encore plus performants, car il est encore loin d'égaliser les majors du domaine. Notre objectif principal n'était pas de développer le logiciel le plus performant du marché, mais plutôt de réaliser un programme propre et fonctionnel, utilisant en priorité les concepts étudiés en cours.

## CONCLUSION

Au final, notre programme répond pleinement aux objectifs initiaux que nous nous étions fixés, à savoir réaliser un générateur de dictionnaire d'empreinte, multi-dictionnaire et multi-supports.

En outre, celui-ci nous a permis d'utiliser dans des cas concrets des concepts de programmation Unix étudiés en cours, à savoir la programmation parallèle, la gestion des signaux, des processus et des mécanismes de communication/synchronisation.

La programmation multithread a été tout particulièrement intéressante, car il s'agit un concept moderne et de plus en plus utilisé grâce à l'avènement architectures multi-cœurs. Par exemple de nombreux jeux commencent à tirer avantageusement parti de 2, 4 voire 8 cœurs. Nous avons aussi été confrontés aux problèmes engendrés par la programmation multitâche : complexité de mise en œuvre, débogage ardu, erreurs d'accès mémoires, etc.

8e105cee0ab7650cc3a3751bd986dbf5

---

Dossier réalisé par **Julien Lafont** ( [www.Studio-Dev.fr](http://www.Studio-Dev.fr) ) dans le cadre de ma formation d'Ingénieur EPSI.

Développement d'une application multi-thread en C++, en utilisant une gestion de projet Agile : Scrum.