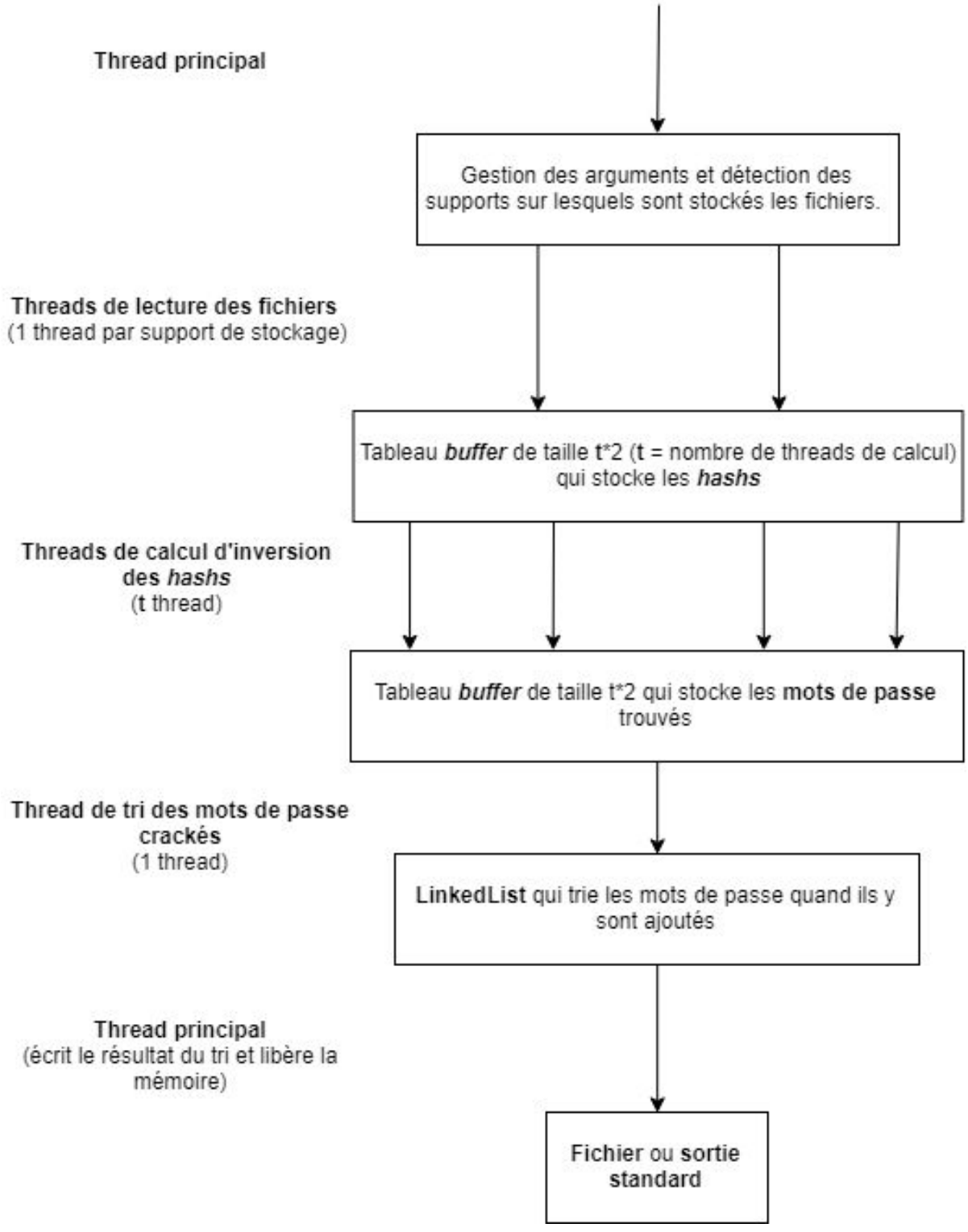
SINF1252 - Password cracker

LAUNOIS Julien
29591700

Architecture haut-niveau du programme



Les *hashs* sont lus et stockés dans un *buffer* par les threads producteurs.
Le thread qui *cracke* les *hashs* est un consommateur de ce *buffer* et également un producteur pour le *buffer* où sont stockés les mots de passe.
Un autre thread consommateur, va ajouter ces mots de passes au fur et à mesure dans une *linked list* tout en faisant le tri en fonction du critère de sélection.
Enfin, lorsque tous les autres threads sont terminés, le thread principal écrit les mots de passe retenus dans un fichier, si celui-ci a été spécifié dans les arguments, ou dans la sortie.

Choix de conception

J'ai décidé de créer ma propre structure (*FilesLocation***)** pour caractériser un support de stockage. Après avoir lu les arguments et récupéré la liste des fichiers, j'initialise cette structure qui comporte un **identifiant**, un **thread** pour la lecture des fichiers, un **entier** qui stocke le nombre de fichiers se trouvant dans ce support, un **boolean** qui sera activé seulement quand le thread de lecture correspondant aura fini son travail, et un **tableau** de chaîne de caractères qui stocke la liste des fichiers.
Au début, le programme trie les fichiers par supports et créer un tableau de structures *FilesLocation*. Ce dernier comportera donc autant d'éléments qu'il y a de supports de stockage différent.
Il stocke ensuite les chemins de chaque fichier dans la liste du support correspondant et enfin, lance les threads de chacun.

Le reste du programme est une implémentation assez fidèle de l'architecture demandée, laquelle est décrite dans la section précédente.

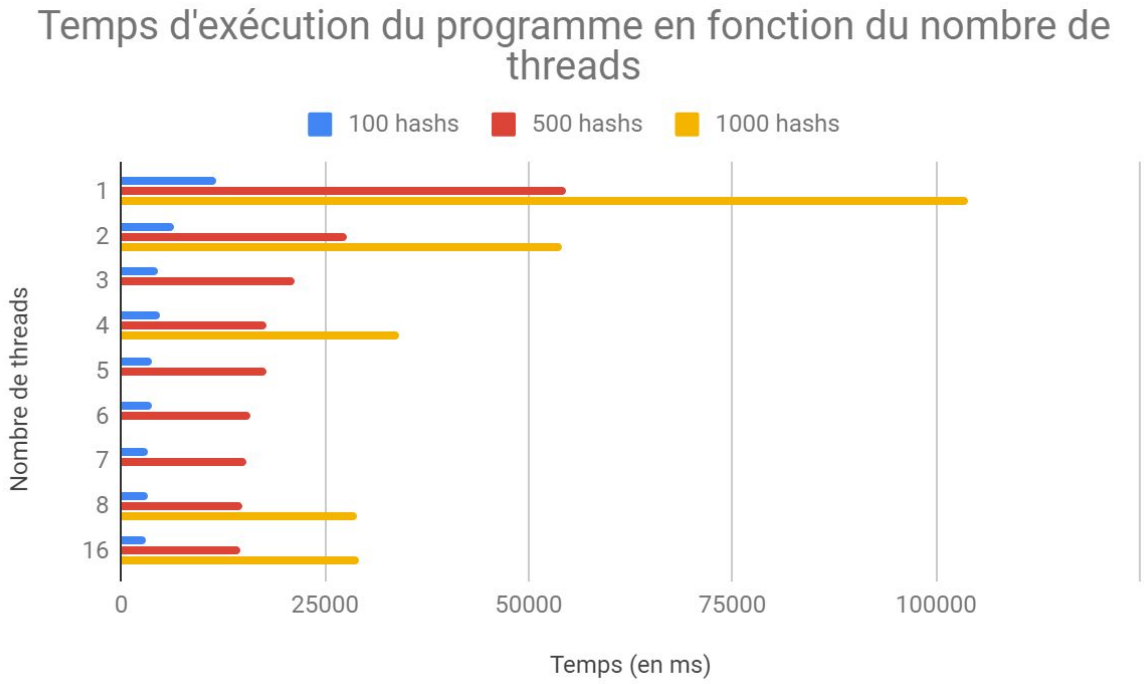
Stratégie de tests

J'ai commencé par faire 3 tests **CUnit** :

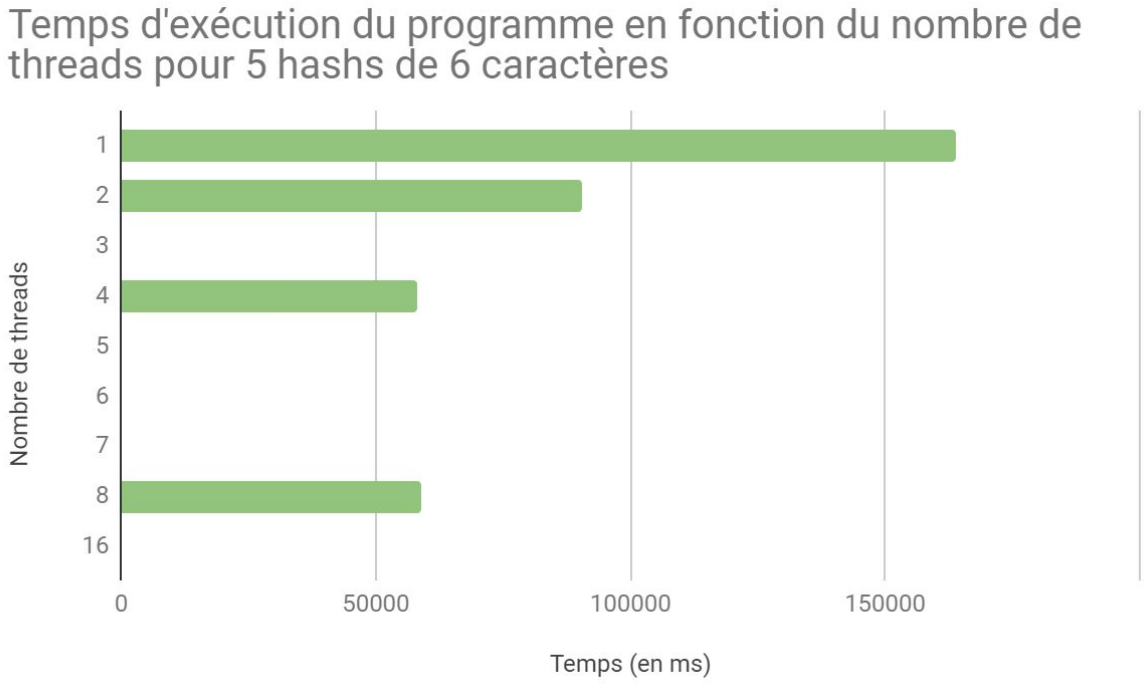
- la fonction **unsigned int getScore(const char *str, const int selection)** qui calcule le nombre de voyelles ou de consonnes dans une chaîne de caractères
- la **linked list,** je teste si elle trie et ajoute bien les mots de passe que je lui envoie à l'aide de la fonction **int addToList(const char *value, const unsigned int score)**
- le **bufferRes** qui est le tableau contenant les mots de passe venant d'être *crackés.* Je teste donc si elle les ajoute bien avec **void insertInBufferRes(char *pswd)** et si elle les retire bien avec **int removeFromBufferRes(char * pswd)**

J'appelle ensuite le programme quelques fois avec différents arguments pour tester si le programme fonctionne bien avec tous types de paramètres. Avant chaque test, il y a une petite explication de ce qui est passé en arguments.

Évaluation quantitative



Pour ces tests, j'ai mesuré le temps que le programme prenait depuis la créations des threads jusqu'à leurs fins. Le fichier utilisé est "01_4c_1k.bin" qui comporte des *hashs* dont les mots de passe correspondant ne dépasse pas 4 caractères. Chaque barre représente une exécution du programme avec une portion plus ou moins grande de ce fichier (respectivement 1/10 puis ½ et enfin 1).



On constate sur ces deux graphes que lorsqu'on passe de 1 à 2 threads, la vitesse est pratiquement doublée. On peut aussi observer qu'au delà de 4 threads, elle ne diminue pratiquement plus, peu importe le nombre de *hashs.* Ceci est probablement dû au fait que les tests ont été faits sur un processeur de 4 coeurs. Néanmoins, le temps d'exécution n'est pas divisé par 4 lorsqu'on atteint ce seuil.