LSINF1252 - Projet Threads - Password Cracker - Rapport

Serena Lucca 29541700 - Damien Van Meerbeeck 28231600

Juin 2019

Ce document est le rapport du Projet Threads du cours LSINF1252. Ce projet consiste en en la création d'une implémentation basée sur le parallélisme par l'utilisation de threads. Le but de l'exécutable est la traduction et la sélection de mots de passes hashés contenus dans un fichier.
Tout d'abord nous avons conçu l'architecture de l'exécutable et ensuite l'implémentation de celui-ci en langage c. Afin de vérifier l'exactitude des résultats rendus par l'exécutable nous avons produit une série de tests unitaires.

1 Architecture haut-niveau du programme

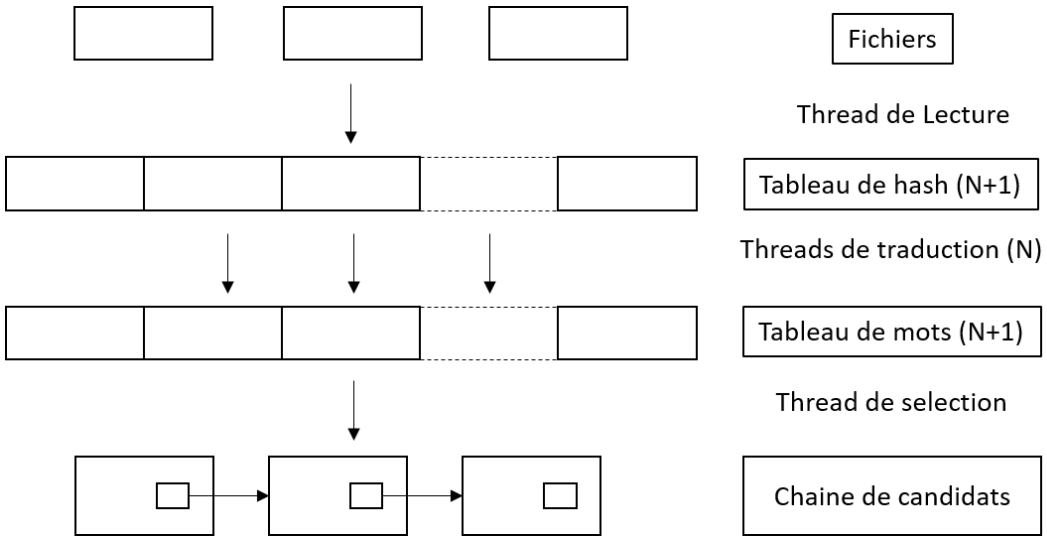

Figure 1: Architecture de l'exécutable cracker

- Les fichiers sont regroupés dans un tableau de pointeurs de taille (nombre de fichiers +1) dont le premier élément correspond au nombre de fichiers et les autres éléments sont des pointeurs vers les noms de fichiers.
- Le tableau hash est un tableau contenant N+1 pointeurs vers une valeur uint8_t (N valant le nombre de threads de traduction). Chaque pointeur vaut soit la valeur NULL soit pointe vers une zone de 32 bytes allouée sur le heap. Chaque zone pointant vers un hash a traduire.
- Le tableau de mots correspond au même tableau précèdent excepté que les pointeurs pointent vers une zone de 16 Bytes et de valeur char. Chaque zone pointant vers un mot traduit.
- La chaîne de candidat est une structure chaînée allouée sur le heap dont chaque maillon possède un pointeur vers un char et un pointeur vers le maillon suivant. Chaque pointeur char pointe vers une zone allouée sur le heap et correspond à un candidat potentiel.
- Le thread de lecture est un producteur. Il prend un pointeur dans le tableau contenant le nom des fichiers, il l'ouvre si possible, lit les hash un par un et place un pointeur vers un hash dans chaque case du tableau de hash.
- Les threads de traduction sont des transformateur. Ils prennent un pointeur dans le tableau de hash, produisent un nouveau pointeur contentant la traduction du hash et le placent dans de tableau de mots.

- Le thread de sélection est un consommateur. Il prend un pointeur dans le tableau de mots, l'analyse pour voir si c'est un candidat puis soit
  - ajoute le mot dans la chaîne
  - élimine la chaîne et place le mot dans une nouvelle chaîne
  - passe au mot suivant

  Une fois le tableau de mots vidé il s'occupe de montrer le résultat à l'utilisateur par la sortie standard ou un fichier de sortie. Ensuite il élimine la chaîne restante et se ferme.

2 Choix de conception

2.1 communication entre threads

La communication entre threads est effectuée par l'utilisation de de variables globales munies d'un mutex et de sémaphores. Les deux tableaux de hash et de mots sont concernés. Pour éviter tout souci de copie double ou de perte d'information chaque threads doit réserver le tableau avant de le modifier grâce au mutex. Et pour éviter de réserver un tableau inutilement, car soit rempli soit vide, le thread regarde si il y a de la place soit pour prendre ou mettre un pointeur, à l'aide de sémaphores.

2.2 Allocation et libération sur le heap

- Les deux tableaux sont initialisé comme variable globale et alloués dans la fonction principale avant la création des threads. Il sont ensuite libérés à la fin de cette fonction.
- Chaque pointeur hash est alloué lors de la lecture et libéré dans le thread de traduction après la traduction reverse-hash.
- Les pointeurs de mots sont alloués dans le thread de traduction et libérés dans la sélection de candidats si ils ne représentent pas un candidat ou quand la liste est éliminé.
- Les maillon sont une structure allouée sur le heap et sont libérée lorsqu'un candidat avec plus d'occurrence est trouvé ou que le thread de sélection est terminé.

3 Stratégie de test

Nous avons décidé de tester le bon fonctionnement des fonctions en lien avec la structure chaînée ainsi que la fonction de sélection des candidats à l'aide de tests unitaires. Nous n'avons pas rédigé de tests avec Cunit par manque de temps et de ressources.

4 Évaluation quantitative

Comme on peut l'observer sur le graphique ci-dessous, le temps d'exécution de notre programme diminue avec l'augmentation du nombre de threads, on atteint un seuil quand le nombre de threads dépasse le nombre de coeurs de l'ordinateur. Cette diminution du temps d'exécution prouve que la parallélisation entre les tâches s'effectue correctement.

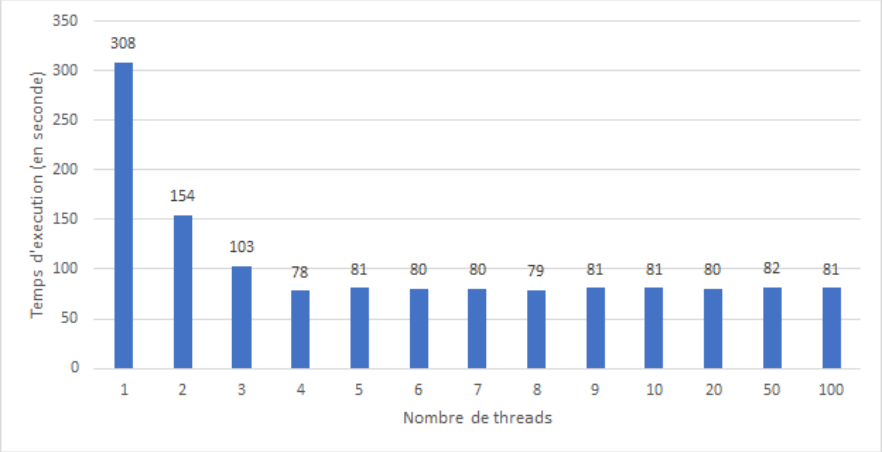

Figure 2: Temps d'exécution de l'exécutable cracker en fonction du nombre de thread