ÉCOLE POLYTECHNIQUE DE LOUVAIN

UCLouvain

Systèmes Informatiques I

LSINF1252

Password Cracker : Architecture du programme

Groupe n°105

Amadéo David - 44761700
Eduardo Vannini - 10301700

1er avril 2019

1 Structures de données

Notre rapport comportera différentes structures de données, à savoir :

- Les **mots de passe non-traités**. Ceux-ci sont représentés par un tableau de trente-deux entiers sous la forme *uint8_t* car un hash est encodé sur trente-deux bytes. Plus simplement, l'énoncé indique aussi que l'argument demandé par la fonction *reversehash* doit être de la forme citée précédemment.
- L'**ensemble des mots de passe non-traités**. Ceux-ci seront stockés dans un buffer (un tableau de pointeur vers des *uint8_t*) de taille prédéfinie et rempli de hashes non-traités. Il est à noter que si la taille du fichier à lire excède la capacité du buffer, la partie qui n'a pas pû être insérée sera ajoutée au buffer par la suite. Notons aussi que le buffer est une structure de type *Last In First Out*.
- L'**ensemble des mots de passe constituants les *meilleurs candidats* à un moment donné**. Ceux-ci sont stockés dans une liste chainée dont les données sont des pointeurs vers des tableau de caractères (autrement dit des pointeurs de pointeurs de caractères) qui représentent les versions déchiffrées des mots de passe retenus comme *meilleurs candidats*.
- **Note** : il est intéressant de remarquer qu'aucune structure n'est utilisée pour représenter un mot de passe car il n'est jamais utile d'accéder en même temps à la version codée et décodée d'un mot de passe en même temps. Dès que celui-ci a été traité, sa version codée peut être oubliée. Les mots de passe seront donc soit représentés par des *int8_t password[32]* comme cité précédemment, soit par des *char** de taille inconnue avant exécution du programme (les versions en clair).

2 Types de threads entrant en jeu

Notre programme comportera plusieurs types de threads, à savoir :

1. Le **thread principal**. Il s'agit du thread dans lequel s'exécute la fonction `main` et qui permet de gérer tous les autres threads du programme. Il est donc le parent des threads décrits ci-dessous. Son rôle est de récupérer les arguments dans `argv` afin d'y réagir de manière appropriée, de créer les diverses ressources partagées ainsi que les mécanismes de protection de celles-ci (mutex, sémaphores, etc), de lancer et de terminer les threads de lecture et de calcul décrits plus bas. Avant que le programme ne termine son exécution, ce thread permet aussi de gérer le "retour vers l'utilisateur", c'est-à-dire qu'il se charge d'informer l'utilisateur du résultat des calculs ;
2. Les **threads de lecture**. Il s'agit de threads **producteurs** qui se chargent de lire le fichier de hashes dans un buffer de taille fixe à déterminer. Il en existe un nombre prédéfini que nous devrons également déterminer à l'aide d'une analyse du remplissage du buffer lors de l'exécution. Ces threads devront être bloqués quand le buffer est rempli et ils se termineront quand le fichier aura été lu dans son entièreté ;
3. Les **threads de calcul**. Il s'agit de threads **consommateurs** dont il existe N instances, avec N une valeur précisée en argument par l'utilisateur et dont la valeur par défaut est 1. Ces threads ont pour rôle d'inverser les hashes présents dans le buffer mentionné ci-dessus, puis de déterminer le "score" du mot de passe obtenu afin de pouvoir mettre à jour la liste des candidats en conséquence. Quand le buffer est vide, ces threads sont bloqués mais ils ne se terminent pas tant que le fichier n'a pas été lu dans son entièreté.

Les fonctions exécutées par nos threads auront un retour de type `int` afin de pouvoir retourner des codes d'erreur en cas de besoin.

3 Méthodes de communication entre les threads

Le programme utilisant le modèle du *producteur-consommateur*, il faudra utiliser deux sémaphores. Le premier, appelé *empty*, représentera le nombre de places libres dans le buffer et sera initialisé à une valeur M, avec M la taille du buffer. Le second sémaphore sera appelé *full*, représentera le nombre de places occupées dans le buffer et sera initialisé à zéro. Enfin, le buffer sera protégé par un mutex.

4 Informations communiquées entre les threads

Au travers de variables partagées situées dans le heap, les threads se communiqueront les informations suivantes :

1. L'**état de lecture du fichier**. Cette information sera communiquée entre les threads de lecture et de calcul. Elle sera décrite par une variable de type `int` dont le nom sera "`file_read`" et dont la valeur sera initialisée à `0` puis mise à `1` dès que le fichier aura été lu en entier. Cette modification de valeur se fera par le thread de lecture ayant détecté la fin du fichier ;
2. L'**indice du dernier hash** inséré dans le buffer. Cette information sera communiquée entre les threads de lecture et de calcul. Elle sera décrite par une variable de type `int` et nommée "`last_hash_index`" qui sera initialisée à la valeur `-1` pour indiquer qu'aucun hash n'a encore été ajouté au buffer. Chaque ajout d'un hash au buffer se fera à la position `last_hash_index + 1` (si celle-ci ne sort pas du buffer) et s'accompagnera de l'incrémentation de `last_hash_index`, tandis que chaque inversion de hash se fera sur celui situé à la position `last_hash_index` (si cette position est positive) et s'accompagnera d'une décrémentation de `last_hash_index` ;
3. Les **hashes du buffer** constituent également des informations que se transmettront entre eux les threads de lecture et de calcul. Nous vous invitons à lire la section *Structures de données* dans laquelle nous décrivons la structure du buffer ;
4. Les **meilleurs candidats** à un instant donné, contenus dans la liste chaînée décrite dans la section *Structures de données*, ainsi que le **meilleur "score"** en ce même instant donné, représenté par une variable "`best_score`" de type `int`, constituent une information échangée par les threads de calcul entre eux, ainsi qu'avec le thread principal.