TP5 : Machine de recherche de mots

1 Introduction

Le but du projet est de mettre en œuvre un moteur de recherche de mots, basé sur un dictionnaire. La machine de recherche de mots possède une interface avec l'utilisateur (fournie dans **main.c**) qui permet de :

1. Charger un fichier dans le dictionnaire. La machine de recherche de mots demande à l'utilisateur de fournir le fichier à charger. L'utilisateur saisit le nom du fichier (par exemple, monfichier.txt) dans la ligne de commande. Ensuite, le moteur de recherche de mots exécute la fonction **add_file**. Une description plus détaillée est fournie dans la section suivante.
2. Recherche un mot dans le dictionnaire. La machine de recherche de mots demande à l'utilisateur de fournir le mot à rechercher. L'utilisateur entre un mot (par exemple, forêt) via la ligne de commande et la machine de recherche de mots exécute la fonction **search_word** et informe l'utilisateur i) dans quels fichiers ce mot existe et ii) combien de fois il a été observé dans chaque fichier.
3. Supprime un fichier du dictionnaire. La machine de recherche de mots demande à l'utilisateur de fournir le fichier à supprimer. L'utilisateur saisit le nom du fichier (par exemple, monfichier.txt) dans la ligne de commande. Ensuite, le moteur de recherche de mots exécute la fonction **remove_file**. Une description plus détaillée est fournie dans la section suivante.
4. Imprimer les mots stockés dans le dictionnaire.
5. Imprimer la liste des fichiers, chargés dans le dictionnaire.

Les codes de ce TP se trouvent dans le répertoire **/share/l3info/CUnix/tp5**. Le dictionnaire qui stocke les mots est implémenté en utilisant une table de hachage avec un chaînage séparé, comme celui vu dans le cours (Figure 1).

Un tableau **filelist** de structures de **listfile_entry** est utilisé pour stocker les noms des fichiers déjà chargés dans le dictionnaire **file** et leur état **loaded**.(Figure 2).

Un ensemble de fichiers de test est fourni dans le dossier **test**.

2 Implementation

Le fichier **types.h** fournit la déclaration des structures requises (**word_entry, word_list, hash_table, listfile_entry**), le nombre maximal de fichiers pouvant être chargés (**MAX_FILES**), la longueur maximale d'un nom de fichier et d'un mot (**MAX_LENGTH**) et la taille de la table de hachage (**MAX_ENTRIES**). Le fichier **functions.h** fournit la déclaration des fonctions. Les fichiers sources et les fonctionnalités sont décrits ci-dessous.

— **main.c** : inclut les fonctions suivantes :

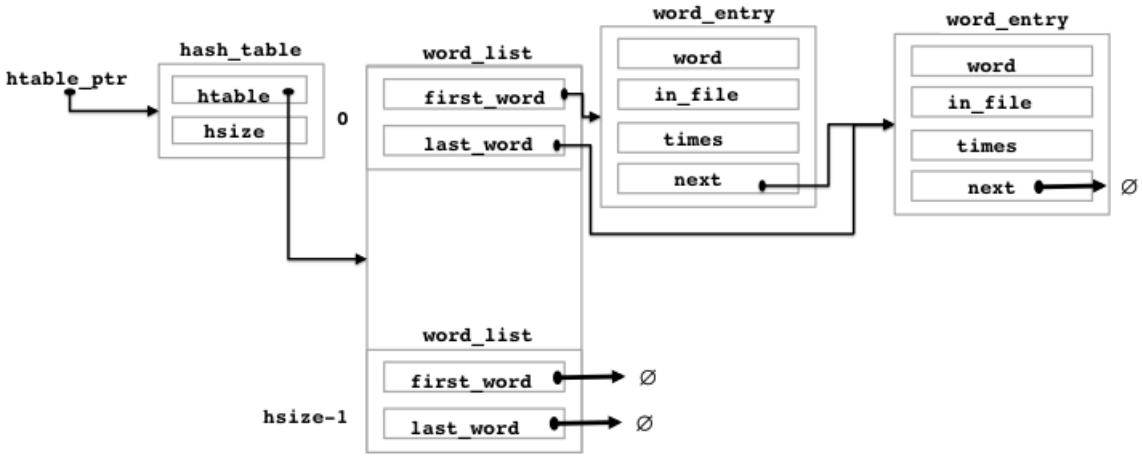

FIGURE 1 – Structure de la machine de recherche avec table de hachage

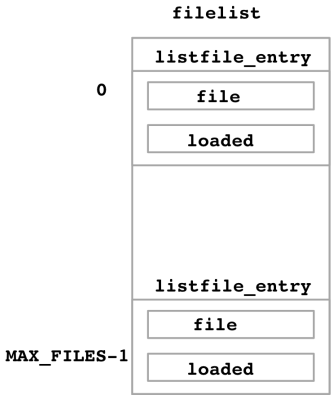

FIGURE 2 – Tableau **filelist** de structures de **listfile_entry**

- **int main()** : affiche le menu des choix de l'utilisateur et gère la création et la suppression des structures de données requises (voir les commentaires dans le code).
- **int hashcode(char word[], int size)** : renvoie la valeur de hachage d'un mot, comme nous l'avons vu pendant le cours. Une fonction typique de hashcode additionne tous les caractères ascii du mot et renvoie l'opération modulo de cette somme et de la taille de la table de hachage.

— **file.c** comprend les fonctions dédiées à la manipulation d'un fichier

- **listfile_entry * create_filelist(int maxfiles)** : crée et initialise la table qui conserve les informations relatives aux fichiers chargés.
- **int add_file(char filename[], listfile_entry * filelist, hash_table * htable_ptr) :** elle vérifie que le fichier à charger (donné par l'utilisateur) n'a pas déjà été chargé. Dans ce cas, elle charge le fichier. Pour cela, le tableau **filelist** est mis à jour en incluant le nom du nouveau fichier à sa première position libre. L'index du tableau est l'identificateur du fichier. Ensuite, elle lit le fichier mot par mot (les chiffres sont exclus) et met à jour

la table en conséquence (voir la fonction **update_table**). Aucune différence ne doit être faite entre les majuscules et les minuscules, par exemple, tous les mots sont stockés en minuscules. Lorsque tous les mots du fichier ont été chargés dans le dictionnaire, l'état du fichier dans le tableau **filelist** est mis à 1, ce qui signifie chargé.

  - **int remove_file(char filename[], listfile_entry * filelist, hash_table * htable_ptr) :** elle vérifie que le fichier à supprimer (donné par l'utilisateur) est chargé dans le dictionnaire. Dans ce cas, elle effectue une recherche dans tout le dictionnaire afin de trouver tous les mots qui appartiennent à ce fichier et les supprime. Si le fichier n'est pas chargé dans le dictionnaire, elle en informe l'utilisateur.
  - **void print_list(listfile_entry * filelist) :** imprime le tableau **filelist** avec les noms des fichiers chargés et leur état.
  - **void free_filelist(listfile_entry * filelist) :** désaffecte la mémoire de la table qui conserve les informations sur les fichiers chargés.
- **hash.c** comprend les fonctions dédiées à la manipulation de la table de hachage :
  - **hash_table * create_table() :** elle crée et initialise la table de hachage.
  - **int search_word(char word[], listfile_entry *filelist, hash_table *htable_ptr) :** elle recherche le mot donné par l'utilisateur dans le dictionnaire et imprime le nombre de fois où il a été trouvé dans chaque fichier. Si le mot n'existe pas, elle en informe l'utilisateur.
  - **void update_table(hash_table * htable_ptr, char word[], char filename[], int file_index)** : elle recherche un mot et met à jour la table en conséquence. Pour un mot, elle vérifie si c'est la première fois que le mot est trouvé dans ce fichier. Dans ce cas, elle ajoute le mot à la liste liée à la position de la table de hachage donnée par la fonction hashcode, en créant une nouvelle **word_entry**. Le mot est stocké dans la table **word**, le nombre d'occurrences (**times**) est initialisé à 1 et l'identificateur du fichier **in_file** est fixé à l'index correspondant du tableau **filelist**. Si le mot a déjà été trouvé auparavant dans ce fichier, il a déjà été inséré dans la liste liée (même mot, même fichier). Dans ce cas, seul le nombre d'occurrences est augmenté d'une unité.
  - **void print_table(hash_table *htable_ptr, listfile_entry * filelist) :** elle imprime les positions non vides du dictionnaire.
  - **void free_table(hash_table * htable_ptr) :** désaffecte la mémoire utilisée pour la table de hachage.

3 Exercice

1. Implémentez les fonctions de la machine de recherche de mots dans les fichiers **main.c**, **file.c** et **hash.c**
2. Complétez le **Makefile** afin de compiler correctement vos fichiers de compilation.
3. Vérifiez le bon fonctionnement et l'implémentation de votre programme (fuite de mémoire, valeurs non initialisées etc.) en utilisant Valgrind.
4. Vous êtes autorisé (et incité) à **ajouter des sous-fonctions supplémentaires** à votre mise en œuvre.

4 Tips

* Programmez progressivement et testez à chaque fois
* N'hésitez pas à insérer des fonctions supplémentaires

4.1 Première étape

— hash.c
  — create_table : crée et initialise une table de hachage vide
  — update_table : ajoute un mot en tête de liste
  — print_table : parcourt la table et l'imprime (sans le nom du fichier, seulement l'index du fichier)
— main.c
  — programmer la fonction hashcode
  — ensuite, tester :
    — create_table,
    — update_table avec quelques valeurs
    — print_table

4.2 Prochaines étapes

Après avoir implémenté et testé la première étape, nous pouvons compléter et tester petit à petit notre programme.

— update_table : compléter la fonctionnalité, ainsi que print_table, et tester différents cas, tels que :
  — mot manquant
  — mot qui se trouve deux fois dans le même fichier
  — même mot qui se trouve dans deux fichiers différents
— search_word : programmer et tester en considérant différents cas, similaire à update_table.
— release_table : programmee et tester avec valgrind

5 Optionnel

Afin de faciliter la recherche, il devrait être possible pour l'utilisateur d'entrer des expressions logiques (au moins par l'opérateur AND), c'est-à-dire mot1 AND mot2 qui retournera les fichiers contenant les deux mots. D'autres opérateurs (par exemple OR, NOT) et le support des parenthèses sont souhaitables !