Projet

1 Introduction

L'objectif de ce projet est de créer une bibliothèque de fonctions pour accéder à une base de données spécialisée pour stocker des couples <*clef*, *valeur*>, où les clefs et les valeurs sont des suites quelconques d'octets, puis de rechercher les paramètres optimaux.

2 Structure de la base de données

L'objectif de la base est de permettre un accès rapide quelle que soit sa taille. On propose donc d'indexer les clefs au moyen d'une table de hachage. On évitera au maximum les accès au disque : si un couple <*clef*, *valeur*> est supprimé au milieu de la base, on préfère conserver l'information d'un espace inoccupé plutôt que décaler tout le reste du contenu.

2.1 Les différents fichiers

Pour implémenter ces principes, on propose de structurer la base en 4 fichiers, comme représenté sur la figure 1.

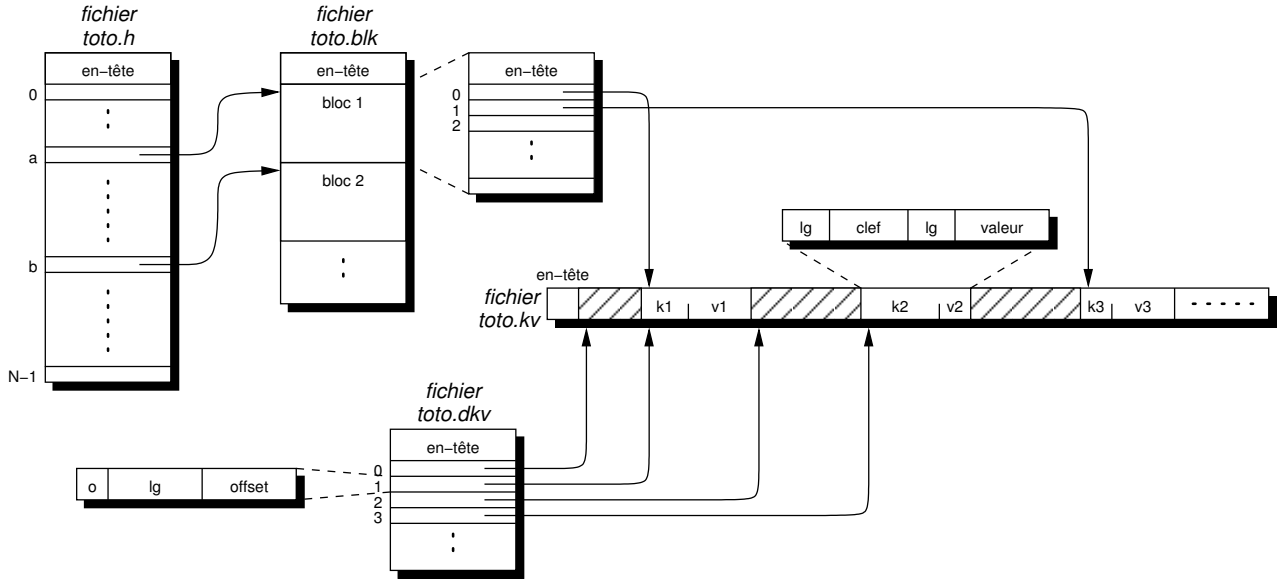

FIGURE 1 – Structure de la base de données

Au début de chaque fichier se trouve un en-tête. A minima, cet en-tête doit contenir un « nombre magique » (une valeur différente pour chacun des 4 fichiers) dont le but est de permettre de vérifier qu'on ouvre un fichier du bon type. Selon les besoins que vous rencontrerez, vous pourrez faire figurer d'autres informations dans ces en-têtes. Pour des raisons pratiques, on limitera les fichiers au maximum à 4 Go (offset sur 32 bits).

Le fichier « `.h` » contient la table de hachage proprement dite. À la création, ce fichier ne doit contenir qu'un en-tête. Lors de l'insertion d'une donnée, on écrira directement l'entrée à l'offset correspondant à la valeur de hachage. On se rappellera que l'utilisation de `lseek` permet d'avoir des fichiers contenant des « trous », c'est-à-dire des blocs non écrits sur le disque et n'occupant donc pas d'espace superflu. Ceci permet d'avoir une très grande valeur de N (nombre maximum d'entrées dans la table de hachage).

Pour gérer les collisions, une entrée dans la table de hachage contient un numéro de bloc $b \geq 1$ [1] dans le fichier

1. La valeur $b = 0$ est réservée à une entrée vide dans la table de hachage.

« `.blk` ». Le bloc est lu en mémoire et les différentes entrées qu'il contient sont parcourues pour trouver la clef recherchée. On considérera qu'un bloc occupe 4096 octets, soit un nombre d'entrées égal à (4096 - taille de l'en-tête du bloc) / 4 octets. Si ce nombre d'entrées s'avère insuffisant, l'en-tête du bloc doit contenir le numéro d'un bloc suivant, ce schéma pouvant être répété autant de fois que nécessaire.

Une entrée dans un bloc contient un offset [2] dans le fichier « `.kv` » contenant les couples <*clef*, *valeur*>. Chaque élément du couple est précédé de sa longueur.

Enfin, le fichier « `.dkv` » contient les descripteurs des emplacements dans le fichier « `.kv` ». Chaque descripteur contient : un bit indiquant si l'emplacement est vide ou occupé par un couple <*clef*, *valeur*> valide, la longueur totale de l'emplacement, et son offset. Pour simplifier, on supposera que l'ensemble de ce fichier, dont la taille dépend uniquement du nombre d'emplacements dans le fichier « `.kv` » peut tenir entièrement en mémoire.

2.2 Principales opérations

Les principales fonctions d'accès à la base sont `kv_put` pour stocker un nouveau couple <*clef*, *valeur*>, `kv_get` pour chercher une clef et la valeur qui lui est associée et `kv_del` pour supprimer un couple associé à une clef.

Pour stocker une nouvelle clef, `kv_put` doit hacher la clef, parcourir les différents blocs référencés par la valeur hachée pour vérifier que la clef n'existe pas déjà, puis trouver une entrée libre dans un de ces blocs. Ensuite, un emplacement libre doit être alloué pour stocker le couple <*clef*, *valeur*> et cet emplacement doit être écrit dans le bloc. Si l'emplacement trouvé est suffisamment grand pour stocker au moins un autre couple (le couple de taille minimum est celui formé d'une clef d'un octet et d'une valeur vide), l'allocation de l'emplacement provoque l'insertion d'un nouvel emplacement vide situé juste après.

Pour supprimer une clef existante, le même parcours (hachage, puis recherche de la clef dans les blocs) est effectué. Lorsque l'emplacement est trouvé, il doit être désalloué, c'est-à-dire que le descripteur correspondant doit indiquer que l'emplacement est vide. Si l'un ou les deux emplacements adjacents est lui-même vide, il faut les fusionner en un seul. Pour simplifier, on ne désallouera pas les blocs vides dans le fichier « `.blk` ».

Si une nouvelle valeur doit remplacer celle d'une clef déjà existante, on pourra utiliser la stratégie simple consistant à supprimer l'ancien couple <*clef*, *valeur*> et ajouter le nouveau.

2.3 Choix de la fonction de hachage

Le travail demandé nécessitant la comparaison de plusieurs fonctions de hachage, on pourra choisir la fonction à utiliser en donnant son numéro lors de la création de la base. Par défaut, on implémentera la fonction de hachage d'index 0 consistant en la somme des octets de la clef, modulo 999983 (plus grand nombre premier inférieur à 1 million).

2.4 Choix de la méthode d'allocation

L'allocation d'un emplacement pour un couple <*clef*, *valeur*> dans le fichier « `.kv` » nécessite une méthode d'allocation. Trois méthodes doivent être définies :

- *first fit* : choisir le premier emplacement disponible suffisamment grand ;
- *worst fit* : choisir l'emplacement disponible le plus grand ;
- *best fit* : choisir l'emplacement disponible le plus petit parmi les emplacements suffisamment grands.

3 Interface de la bibliothèque

L'utilisation des fonctions de la bibliothèque nécessite l'inclusion du fichier `kv.h` (fourni sur Moodle). Celui-ci contient :

- une définition du type `KV`, le descripteur renvoyé lors de l'ouverture de la base et utilisé par toutes les autres fonctions ;
- une définition du type `kv_datum` utilisé pour représenter une clef ou une valeur ; un élément de ce type contient une longueur (en octets) et un pointeur vers les données elles-mêmes ;

2. Là encore, la valeur 0 indique que l'entrée est vide.

— une définition des différentes méthodes d'allocation (voir plus loin) ;
— les prototypes des fonctions de la bibliothèque.

Les fonctions qu'on vous demande d'écrire étant censées être utilisées par des applications, elles ne doivent rien afficher et ne comporter aucun effet de bord autre que ceux prévus (modifier la base) ou implicites (allouer de la mémoire) ; en particulier, il est hors de question d'afficher des messages. Lorsqu'une erreur est rencontrée, les fonctions doivent positionner la variable `errno` pour indiquer la raison de l'erreur (ou une approximation) et renvoyer un code approprié (`-1` ou `NULL`) suivant le type de retour de la fonction.

Les fonctions que vous devez rédiger sont :

— `KV *kv_open (const char *dbname, const char *mode, int hidx, alloc_t alloc)`
Cette fonction ouvre la base dont les fichiers commencent par la chaîne indiquée par `dbname`. Le mode d'ouverture peut être « `r` », « `r+` », « `w` » ou « `w+` » (voir `fopen` pour la signification). Le paramètre `hidx` donne l'index de la fonction de hachage à utiliser avec cette base, la valeur 0 correspondant à la fonction par défaut ; ce paramètre n'a de signification que lors de la création de la base. Enfin, le paramètre `alloc` contient la méthode d'allocation. La valeur renvoyée est le descripteur d'ouverture de base utilisé par toutes les autres fonctions.
— `int kv_close (KV *kv)`
Cette fonction termine l'accès à la base.
— `int kv_get (KV *kv, const kv_datum *key, kv_datum *val)`
Cette fonction cherche la clef indiquée par `key`. Si elle est trouvée, elle renvoie `1` ainsi que la valeur dans `val`. Si le pointeur inclus dans `val` est nul, `kv_get` alloue la mémoire nécessaire ; sinon, la longueur incluse dans `val` précise la taille maximum des données que `kv_get` peut stocker à l'adresse indiquée. Dans tous les cas, la longueur en sortie contient le nombre d'octets transférés dans `val`. Enfin, si la clef n'est pas trouvée, la fonction renvoie `0`.
— `int kv_put (KV *kv, const kv_datum *key, const kv_datum *val)`
Cette fonction stocke le couple <*clef*, *valeur*> dans la base. Si la clef y figurait déjà, la nouvelle valeur remplace l'ancienne.
— `int kv_del (KV *kv, const kv_datum *key)`
Cette fonction supprime le couple <*clef*, *valeur*> de la base. Si la suppression s'effectue correctement, la fonction renvoie `0`. La tentative de suppression d'une clef inexistante est considérée comme une erreur (avec `errno` = `ENOENT`).
— `void kv_start (KV *kv)`
Cette fonction prépare les appels ultérieurs à `kv_next` pour balayer la liste des couples <*clef*, *valeur*> présents dans la base.
— `int kv_next (KV *kv, kv_datum *key, kv_datum *val)`
Cette fonction est utilisée pour balayer la liste des couples présents dans la base. Chaque appel renvoie la valeur `1` ainsi que le couple suivant dans `key` et `val` (dont les pointeurs peuvent être nuls, comme pour `kv_get`). Lorsqu'il n'y a plus de couple, `kv_next` renvoie `0`.

Tous les accès aux fichiers effectués par ces fonctions doivent être réalisés en utilisant exclusivement les primitives systèmes `open`, `close`, `read`, `write`, `lseek`, `fstat` et `ftruncate`. Les fonctions de bibliothèque non liées aux fichiers peuvent également être utilisées (allocation mémoire, recopie de mémoire, etc.).

4 Éléments fournis

Vous trouverez sur Moodle une archive contenant plusieurs éléments :

— un fichier `kv.h` prêt à l'emploi : vous ne devez pas le modifier puisqu'il est conforme aux spécifications décrites ci-dessus ;
— des fichiers sources (`get.c`, `put.c`, `del.c` et `common.*`) montrant des exemples d'utilisation des fonctions demandées ;
— des scripts de test (`test-`*`nnn`*`.sh`) utilisant les exemples ci-dessus pour automatiser les tests, que vous pouvez compléter en fonction de votre implémentation ;
— un fichier `Makefile` pour automatiser un certain nombre d'actions.

5 Travail demandé

On demande de réaliser :

— la bibliothèque de fonctions, sous forme d'un fichier « `kv.c` » contenant les fonctions demandées ;
— des jeux de tests complémentaires et la mesure de la couverture de code, pour vérifier que vos fonctions sont bien conformes aux spécifications ;
— deux (au moins) fonctions de hachage permettant d'obtenir de meilleures performances que la fonction par défaut, avec les mesures associées ;
— des mesures pertinentes et reproductibles des différentes méthodes d'allocation pour déterminer la méthode optimale ;
— un rapport décrivant les élements demandés précédemment.

Votre implémentation doit fonctionner sur `turing`. Vous vous attacherez à respecter les spécifications et à le prouver par vos jeux de tests, à veiller à ne pas avoir de fuite de mémoire et à effectuer des mesures reproductibles et significatives.

6 Modalités de remise

Le projet est à réaliser par groupes de deux étudiants. Votre projet doit contenir :

— les fichiers sources ;
— le rapport au format PDF, contenant en particulier les mesures demandées ;
— les scripts utilisés pour reproduire vos diverses mesures ;
— l'ensemble des jeux de tests et la mesure de couverture obtenue avec `gcov`.

Vous déposerez votre projet, débarrassé de tout fichier binaire autre que votre rapport, sous forme d'une archive au format *tar.gz* sur Moodle dans l'espace prévu à cet effet, avant le lundi 4 avril 2016 à 12h.