

Key-value Store: Project 2018

Andrea RAR, Ryan SIOW, Jonas EPPER

IN.4022 Operating System Course, University of Fribourg
andrea.rar@unifr.ch, ryan.siow@unifr.ch, jonas.epper@unifr.ch

Table des matières

1	Introduction	1
2	Description du problème	1
3	Solutions et décisions prises	1
3.1	KV store: array dynamique de struct	2
3.2	communication server - clients	3
3.3	lire,écrire et modifier des valeurs	3
3.4	Accès sécurisé des lecteurs/rédacteurs	3
3.5	scripts de tests	3
4	apprentissage	3
5	Conclusion	3
A	Manuel d'utilisateur	3

1 Introduction

Tous. rester TRES COURT et CONCIS

2 Description du problème

Tous. décrire le problème en gros, on ira en profondeur plus tard

Plusieurs problèmes devaient être traités durant ce projet. Nous avons aussi dû prendre des décisions d'implémentation qui rendait notre solution unique et si possible proche de l'efficacité optimale.

La première étape était de créer une structure permettant de recevoir des paires de clés et de valeurs. Cette structure devait être dynamique, c'est-à-dire grandir à mesure que l'on ajoutait des paires afin d'utiliser le moins de mémoire possible. Cette structure devait supporter des actions telles que ajouter, modifier, lire ou supprimer des paires.

De plus, il fallait gérer une communication entre le serveur et ses clients. le serveur devait être transparent pour les clients et ceux-ci devait juste pouvoir envoyer des requêtes simples et le serveur leurs retournait les résultats des requêtes.

Finalement, les accès à la structure de données devait être sécurisés afin qu'aucun conflit d'occure. Il ne faudrait par exemple pas que deux clients modifie en même temps la même pair, ceci pouvant générer des comportements inattendus et imprévisibles.

3 Solutions et décisions prises

Dans cette partie, nous allons décrire les solutions que nous avons élaboré pour résoudre chaque problème mentionné auparavant.

3.1 KV store: array dynamique de struct

Nous avons opté pour ce projet, de construire un tableau redimensionnable. Les valeurs ainsi que les clés sont ajoutées les unes après les autres dans le tableau. Lorsque les éléments atteignent la taille maximale du tableau qui lui est initialement attribuée, le tableau est redimensionné à deux fois sa taille initiale, et ainsi de suite.

Les fonctions pouvant manipuler le tableau permettent d'ajouter, supprimer, modifier et lire les éléments de ce dernier. Lorsque un élément est supprimé du tableau, il laisse un trou qui peut être complété par la prochaine valeur ajoutée, au lieu que cette dernière s'ajoute à la fin du tableau. Cela permet d'optimiser l'espace (trou complété) en n'allouant le moins possible de mémoire.

Voici la structure du tableau:

```

1  typedef struct{
      int key;
3     char *value;
      size_t used; //indicate where we are in the array
5     size_t size; //indicate the size of the array
}KVstore;
```

Ce tableau dynamique comporte 3 fonctions permettant d'initialiser, d'ajouter des éléments ainsi que de libérer la mémoire allouée.

Bien que ces fonctions méritent d'être étudiées plus en détail, seule une description de "insertKV()" est donnée. En effet, cette fonction est la plus intéressante car c'est dans celle-ci que l'optimisation de l'espace alloué est faite.

Tout d'abord, la fonction redimensionne le tableau si celui-ci en a besoin grâce à la méthode `realloc()`. La valeur "newvalue" alors passée en argument, si elle n'est pas nulle, pourra être insérée dans le tableau, soit dans un espace libre s'il y en a, soit à la fin de la série d'éléments déjà présents. Voici un extrait de code explicite:

```

1  if (newvalue != NULL) {
2     size_t length = strlen(newvalue);
      //check if there are holes, if yes add in the hole
4     for(int j = 0; j<=kv->used; j++){
          if(kv[j].key == -1 || kv[j].value == NULL ){
6             size_t length1 = strlen(kv[j].value);
              kv->used++;
8             //clear the memory in this index
              memset(kv[j].value, 0, length1);
10            // insert new value
              strncpy(kv[j].value, newvalue, length);
12            kv[j].key = newkey; // indicate key
              break;
14        }
          //if no holes, just add at the end of the series
16        if(j == kv->used){
              kv->used++;
18            strncpy(kv[j].value, newvalue, length);
              kv[j].key = newkey; // indicate key
20        }
22    }
}
```

Le tableau redimensionnable a été choisi pour sa simplicité d'implémentation. Toutefois, tel qu'il est implémenté dans ce code, ce tableau comporte un inconvénient: il n'est pas de possibilité de réduire la mémoire allouée (la taille du tableau exacte) pour le tableau si les éléments sont supprimés. Les éléments sont ajoutés dans les trous créés par les éléments supprimés afin de pallier au problème et optimiser l'utilisation de la mémoire. Il est aussi déconseillé d'allouer la mémoire à nouveau lorsque la taille du tableau est réduite, car une copie de ce dernier doit être faite à chaque processus de réduction. Des pertes d'éléments peuvent en être la conséquence. Une meilleure implémentation aurait été une liste liée (linked list), ainsi, le problème de dimensionnement lors d'une suppression d'élément disparaît.

3.2 communication server - clients

nombres de client max Andrea : renvoie des messages aux client Jonas tout le reste

3.3 lire, écrire et modifier des valeurs

Tous Regex,

3.4 Accès sécurisé des lecteurs/rédacteurs

Andrea qui a la priorité? read/write etc nombre de clients max? reader/writer nombres citer la page web pour le problème des lecteurs/rédacteurs

3.5 scripts de tests

Jonas

italique: *exemple italique*

4 apprentissage

ce que nous avons appris Tous

5 Conclusion

References

[1] ??? *titre de l'article*, ref. Read the

[2] ??? *faire un tableau dynamique en C*. <http://www....>, Last visited:

A Manuel d'utilisateur

Le fichier README vous explique de manière détaillée comment compiler et exécuter le programme.