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

Ce mini-projet a pour but d'implémenter un stockage clé-valeure avec des accès simultanés via TCP. Ce type de donnée est appelé Key-Value Store (KV). Le stockage utilise un tableau associatif comme modèle. Dans ce dernier, les données sont représentées comme une collection de pairs tel que chaque clé est unique dans la liste. L'utilisation de sémaphore ou de mutex étudié en cours permet de pallier au problème de "race condition".

2 Description du problème

Plusieurs problèmes doivent être traités durant ce projet. Ce dernier doit permettre d'écrire les clés avec les valeurs via TCP sockets. Lors du processus de lecture, il doit être possible de lire les valeurs via TCP sockets en fournissant la clé (un message d'erreur s'affiche si la clé entrée n'existe pas). Le programme doit utiliser des threads multiples et un accès simultané et sûr doit être garantit aux "readers" et "writers". Enfin, un script permettant un test automatique doit être fournit pour faciliter la tâche de révision du code.
Durant ce projet, diverses décisions d'implémentation ont dû être prise pour rendre la solution unique et proche de l'efficacité optimale.
La première étape consiste à créer une structure permettant de recevoir des paires de clés et de valeurs. Cette dernière doit être dynamique, c-à-d grandir à mesure que des paires clé-valeurs sont ajouté dans le but d'utiliser le moins de mémoire possible. En outre, cette structure doit supporter des actions telles qu'ajouter, modifier, lire ou supprimer des paires.
La deuxième étape se concentre sur la gestion de la communication entre le serveur et ses clients. Le serveur doit être transparent pour les clients, et ceux-ci doivent pouvoir envoyer des requêtes simples au serveur qui se charge de retourner les résultats.
Finalement, les accès à la structure de donnée doit être sécurisé afin qu'aucun conflit ne surgisse. Par exemple, il est interdit que deux clients puissent modifier en même temps la même pair, ceci pouvant générer des comportements inattendus et imprévisibles.

3 Solutions et décisions prises

Dans cette partie, les solutions élaborées pour résoudre chaque problème mentionné auparavant sont décrites:

1. Structure KV store (Ryan)
2. Communication serveur-client (Andrea, Jonas)
3. Fonctions: lire, écrire, modifier, supprimer et Regex (Tous)
4. Accès sécurisé des "readers"/"writers" (Andrea)
5. Script de test (Jonas)

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 élément 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:

```
typedef struct{
    int key;
    char *value;
    size_t used; //indicate where we are in the array
    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 passé 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:

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

Le tableau redimensionable a été choisit 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 alloué (la taille du tableau exacte) pour le tableau si les éléments sont supprimés. Les éléments sont ajouté 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 lié (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

Une fois que la commande a été traitée par le serveur, ce dernier affiche le résultat dans sa propre fenêtre et de plus envoie la réponse au client. Afin d'envoyer la réponse au client, chaque fonction traitant la commande modifie un string global se trouvant dans le serveur. Ainsi une fois que la commande a été entièrement traitée, le string est envoyé en une seule fois au client afin de maximiser les performances. En effet l'envoie de messages est plus lent que la modification d'un string global et le client reçoit ainsi qu'une seule réponse contenant l'intégralité.

3.3 Lire, écrire et modifier des valeurs

Afin de vérifier qu'une commande reçue par le serveur (qu'elle soit envoyée par le client ou écrite directement dans la fenêtre du serveur) corresponde à la structure décrite dans le fichier README.txt, l'utilisation de regex s'est imposée. Si la commande ne correspond pas au regex, elle est tout simplement rejetée. L'utilisateur est informé que sa requête n'est pas valide et il peut en insérer une autre.

Les fonctions permettant de manipuler le KV store grâce aux commandes sont:

```
void addpair(int newkey, char* newvalue);
void deletepair(int key, char* value);
void modifyPair(int key, char* value1, char* value2);
void readpair(int key, char* value);
void printKV();
```

Les trois premières fonctions sont de type "write" et les deux dernières de type "read". En effet, il n'est pas permit d'ajouter, supprimer ou de modifier la même pair par peur d'avoir des comportement imprévisible et ingérable. En revanche, il est tout à fait possible de lire en même temps.

3.4 Accès sécurisé des lecteurs/rédacteurs

Lorsqu'un client veut écrire (writer) dans le Key-Value Store, aucun autre client ne peut écrire en même temps. Cependant, des clients voulant lire (reader) les données stockées, peuvent le faire mais ne peuvent cependant pas lire la donnée qui est sur le point d'être insérée, modifiée ou supprimée. Cette dernière est affiché une fois que la modification est terminée. Lorsque plusieurs clients veulent lire en même temps le Key-Value Store, rien de spécial n'est effectué étant donné qu'aucun ne modifie les entrées. Ainsi, un nombre indéfini de clients peut lire le Key-Value Store en même temps.

Afin de réaliser les fonctionnalités cités plus haut, la tableau est divisé en 2 parties principales, la partie qui n'est pas modifiée actuellement et la partie modifiée en ce moment. La deuxième partie est donc protégé par des mutex afin d'éviter d'être lu lorsqu'un client est en train de la modifier. Afin de délimiter cette seconde partie, il est nécessaire de prendre en compte les index dans le tableau où une clé est ajoutée, modifiée ou supprimée. Ainsi, cette seconde partie est protégée par des mutex.

La solution implémentée utilise 4 mutex différents: rmutex, wmutex, readTry et resource. Ceux-ci sont utilisés dans 3 sections: Entry Section, Critical Section (section où les opérations critiques sont réalisées) et Exit Section. Avant d'entrer dans la Critical Section, chaque client doit entrer dans l'Entry Section. Cependant, il peut y avoir qu'un seul client dans l'Entry Section à la fois. Ceci est fait pour éviter les race condition (ex: deux readers incrémentent le readcount au même moment, et les deux essayent de bloquer la ressource, provoquant le bloquage d'un reader).

De plus, les mutex sont implémenté de façon à ce que les clients en écriture aient la priorité. Explication: si le reader R1 bloque l'accès au writter W et que R2 arrive, il ne doit pas pouvoir passer devant W sinon ce serait injuste et ce dernier mourra. Ainsi, une contrainte est ajoutée afin qu'aucun writer, une fois ajouté à la queue, ne doive attendre plus qu'absolument nécessaire.

Cette contrainte est réalisée en obligeant chaque reader à bloquer et relacher le semaphore readtry individuellement. Le writer au contraire ne doit pas le bloquer individuellement. Seulement le premier writer bloque le readtry et ensuite tous les writers ultérieure peuvent simplement accéder à la resource quand elle est libérée par le writer précédent. Le dernier writer doit relacher le semaphore readtry afin d'ouvrir la possibilité aux readrers d'essayer d'écrire.

De l'autre côté, si un reader a bloqué le semaphore readtry, ceci va indiquer à tous les writers potentiels qu'il y a un reader dans la section d'entrée. Ainsi, le writer va attendre que le reader relâche le readtry et ensuite le writer bloquera immédiatement le readtry.

Le semaphore resource peut être bloquer par le reader ou le writer. Ils peuvent le faire uniquement après avoir bloqué le semaphore readtry, ce qui peut être fait uniquement par l'un d'entre eux à la fois.

Rmutex et wmutex permettent d'éviter les race conditions pour les reader et les writers pendant qu'ils sont dans la section d'entrée et de sortie.

Problèmes de cette solution: dans certains rares cas, elle peut résulter en une "starvation" des readers étant donné qu'ils n'ont pas la priorité. Afin de palier à ce problème, il aurait fallu que l'opération de bloquage des ressources partagées se termine toujours après un certain temps donné. Ainsi il aurait été necessaire de stocker les readers et les writers dans une queue (FIFO) et obliger les semaphore à maintenir cet ordre.

3.5 scripts de tests

Jonas
italique: *exemple italique*

4 apprentissage

Andrea:

Jonas:

Ryan: N'ayant jamais fait de tableau dynamique auparavant, j'ai rencontré plusieurs problèmes lors de la construction de la structure ainsi que de la dynamicité du tableau; spécialement lors de la suppression d'un élément, laissant ainsi un trou dans le tableau. Néanmoins je crois avoir accumulé une certaine expérience non négligeable dans ce domaine. En outre, il est difficile de parfois comprendre ce qu'un autre membre du groupe a codé, cela implique d'avoir une structure de travail avec lequel il faut suivre prudemment.

5 Conclusion

References

[1] *Mutex.* `https://en.wikipedia.org/wiki/Readers%E2%80%93writers_problem`

[2] *Faire un tableau dynamique en C.* `https://stackoverflow.com/questions/3536153/c-dynamically-growing-array?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa`, Last visited: 16.04.2018

A Manuel d'utilisateur

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