

Miniproject MP01: Key-Value Store in Linux

Groupe 3:
Jonas Epper, Andrea Rar, Ryan Siow

Table des matières

- Introduction
- Description du problème
- Solution
 - KV store : array dynamique de struct
 - Communication server - clients
 - Lire, écrire et modifier des valeurs
 - Accès sécurisé des readers/writers
- Problèmes rencontrés
- Manuel d'utilisateur
- Démo
- Conclusion

Introduction

- «Simple In-Memory Key-Value store with simultaneous access via TCP in Linux»
- Data model: associative array
 - «Data is represented as a collection of key-value pairs, such that each possible key appears at most once in the collection»

Description du problème

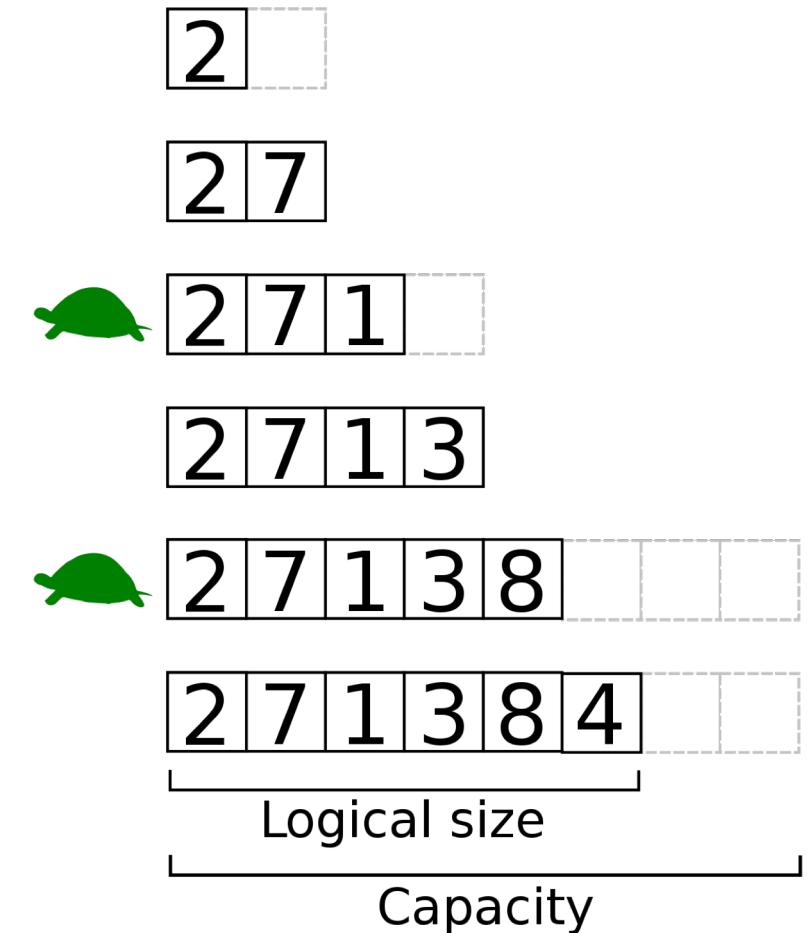
- Minimal Feature Set
 - TCP sockets: lire et écrire
 - Utilisation de plusieurs threads
 - Accès simultané de plusieurs readers et writers
 - Fichiers de test automatisé

Solution

- KV store : array dynamique de struct
- Communication server - clients
- Lire, écrire et modifier des valeurs
- Accès sécurisé des readers/writers

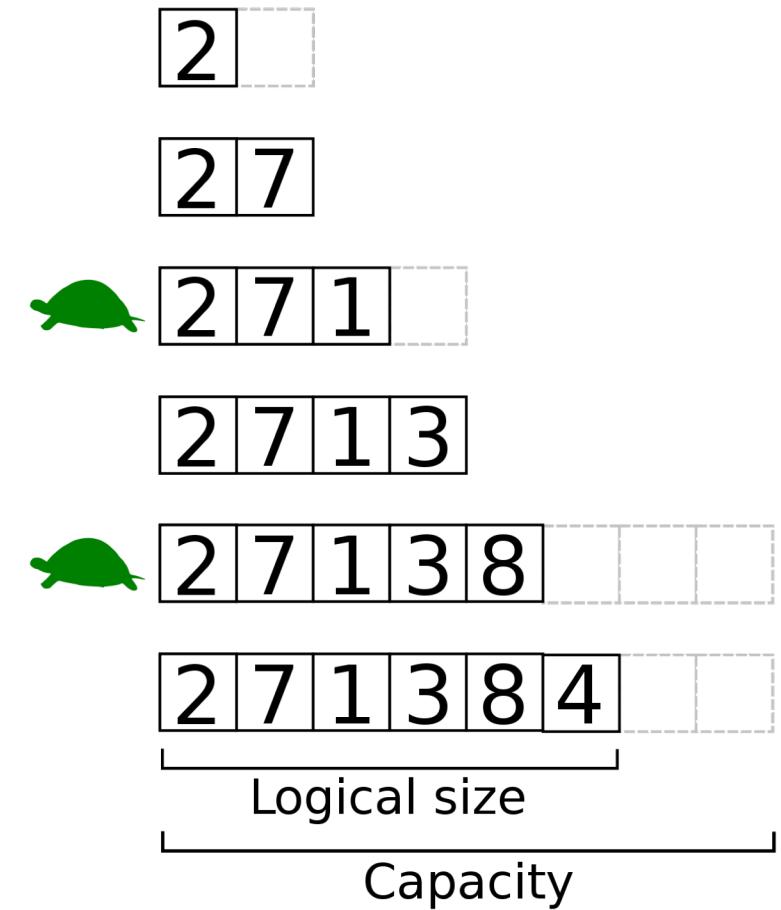
KV store : array dynamique de struct

```
// Key value storage.  
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;
```



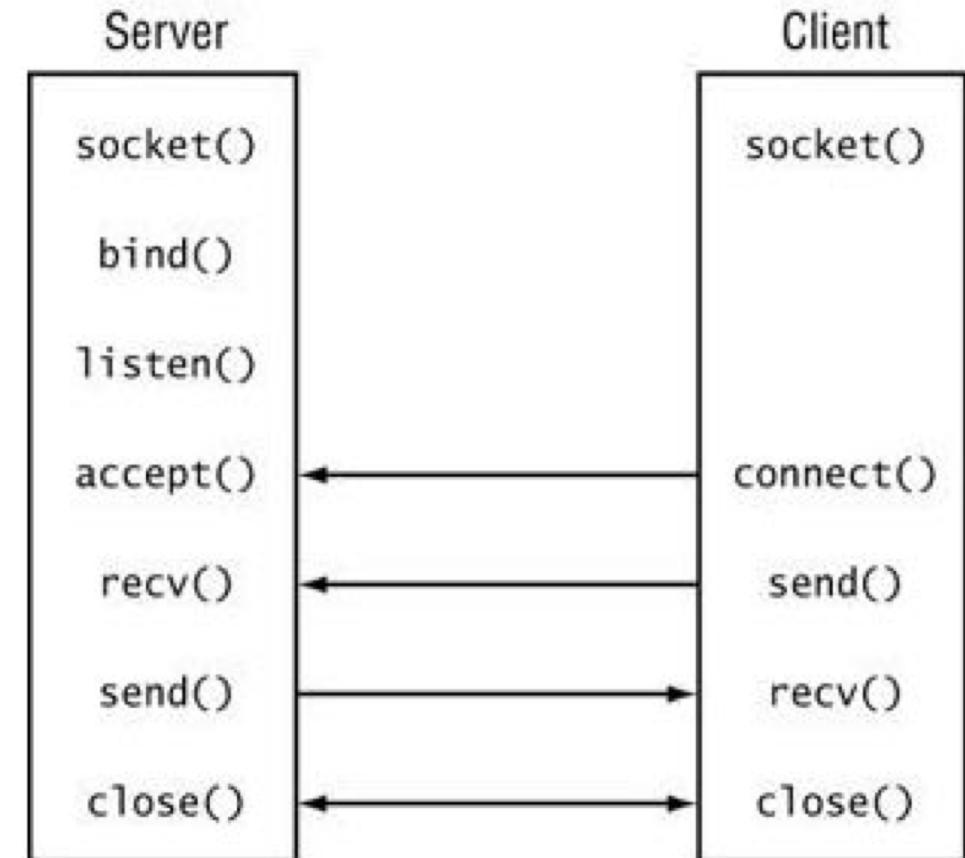
Comparison of list data structures

	Linked list	Array	Dynamic array	Balanced tree	Random access list	Hashed array tree
Indexing	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(\log n)$	$\Theta(\log n)$ ^[10]	$\Theta(1)$
Insert/delete at beginning	$\Theta(1)$	N/A	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)$	$\Theta(n)$
Insert/delete at end	$\Theta(1)$ when last element is known; $\Theta(n)$ when last element is unknown	N/A	$\Theta(1)$ amortized	$\Theta(\log n)$	$\Theta(\log n)$ updating	$\Theta(1)$ amortized
Insert/delete in middle	search time + $\Theta(1)$ ^{[11][12][13]}	N/A	$\Theta(n)$	$\Theta(\log n)$	$\Theta(\log n)$ updating	$\Theta(n)$
Wasted space (average)	$\Theta(n)$	0	$\Theta(n)$ ^[14]	$\Theta(n)$	$\Theta(n)$	$\Theta(\sqrt{n})$



Communication server - clients

- Simple, ordonné, syncro
- Messages renvoyés aux clients



Lire, écrire et modifier des valeurs

- Regex
- Manipulation du Key-Value Store :
 - addpair
 - deletelpair
 - modifyPair
 - readpair
 - printKV

Accès sécurisé des readers/writers

- 4 mutex :
 - rmutex
 - wmutex
 - readTry
 - resource
- 3 zones :
 - Entry Section
 - Critical Section
 - Exit Section
- Priorité au writer
- Nombre de clients max indéfini

Problèmes rencontrés

- Retours à la ligne lors de réception de messages
- Mutex → tableau coupé en plusieurs parties
- Client plus lent que le serveur
- Segmentation fault: 11
- Ainsi qu'une centaine d'autres...

Démo

- Reader(s) arriving while another reader is reading from the KV store
- Writer(s) arriving while another writer is writing in the KV store
- Writer(s) arriving while readers are reading from the KV store
- Reader(s) arriving while a writer is writing in the KV store

Conclusion

La solution présentée ainsi que le code répondent à la consigne, c'est-à-dire aux 5 fonctionnalités minimales. En outre, les décisions d'implémentation concernant la priorité des "readers"/"writters", de la structure de donnée ainsi que des accès simultanés semblent efficaces et optimisés dans leur forme, bien que ces solutions ne soient des plus idéales.

Questions ?

