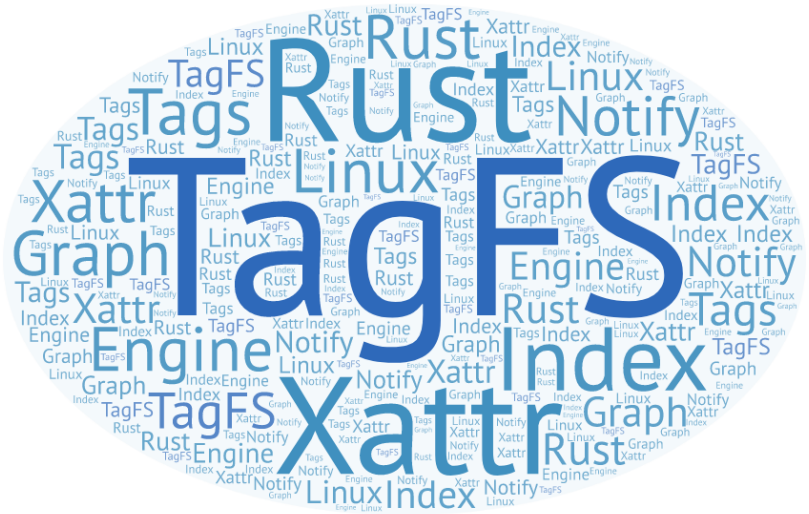
TagFS - Système d'étiquetage des fichiers avec Rust

Steven Liatti

Projet de bachelor - Prof. Florent Glück - Hepia ITI 3ème année

4 septembre 2018



h e p i a
Haute école du paysage, d'ingénierie
et d'architecture de Genève

Hes·SO GENÈVE
Haute Ecole Spécialisée
de Suisse occidentale

Plan

Problématiques

- Nombre de fichiers énorme.

Problématiques

- Nombre de fichiers énorme.
- Difficulté à retrouver des fichiers.

Problématiques

- Nombre de fichiers énorme.
- Difficulté à retrouver des fichiers.
- Plusieurs emplacements logiques pour un seul fichier.

Problématiques

- Nombre de fichiers énorme.
- Difficulté à retrouver des fichiers.
- Plusieurs emplacements logiques pour un seul fichier.

Système de "tagging" de fichiers et répertoires avec possibilité de recherche par tags.

Cahier des charges

- Étudier et s'approprier le langage Rust.

Cahier des charges

- Étudier et s'approprier le langage Rust.
- Répertorier les applications existantes permettant d'étiqueter les fichiers.

Cahier des charges

- Étudier et s'approprier le langage Rust.
- Répertorier les applications existantes permettant d'étiqueter les fichiers.
- Étudier les attributs étendus (XATTR) lors des manipulation courantes sur les fichiers.

Cahier des charges

- Étudier et s'approprier le langage Rust.
- Répertorier les applications existantes permettant d'étiqueter les fichiers.
- Étudier les attributs étendus (XATTR) lors des manipulation courantes sur les fichiers.
- Analyser les moyens d'indexer et de surveiller une arborescence de fichiers.

Cahier des charges

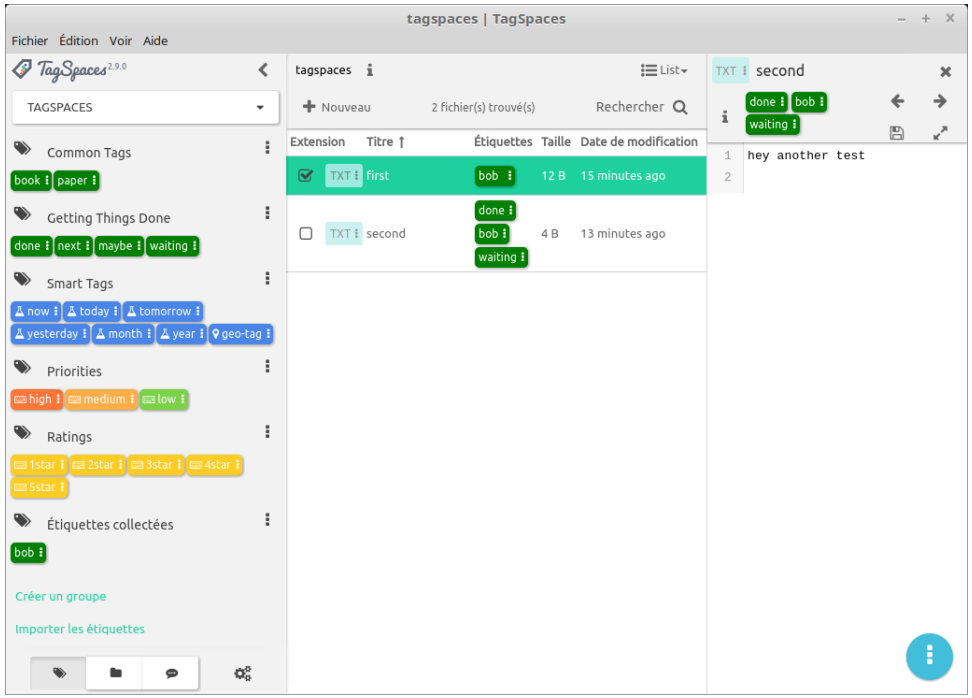
- Étudier et s'approprier le langage Rust.
- Répertorier les applications existantes permettant d'étiqueter les fichiers.
- Étudier les attributs étendus (XATTR) lors des manipulation courantes sur les fichiers.
- Analyser les moyens d'indexer et de surveiller une arborescence de fichiers.
- Concevoir et implémenter le système (open source et sur Linux) et mesurer ses performances.

TMSU, Tagsistant et TagSpaces

- Gestion des tags.
- Liste de fichiers liés aux tags.
- CLI ou GUI.

TMSU, Tagsistant et TagSpaces

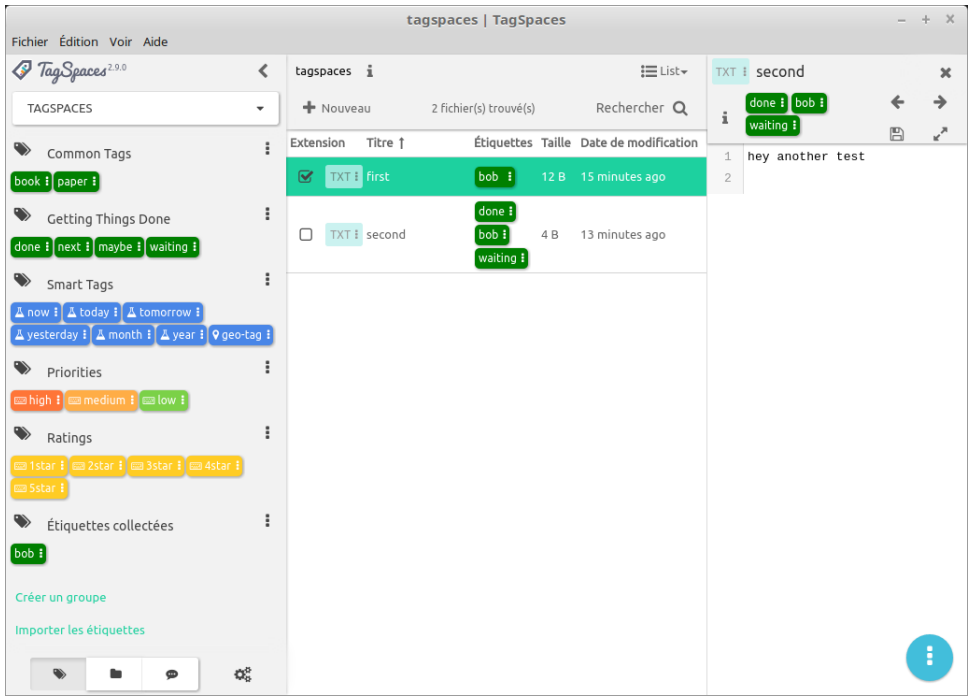
- Gestion des tags.
- Liste de fichiers liés aux tags.
- CLI ou GUI.



TMSU, Tagsistant et TagSpaces

- Gestion des tags.
- Liste de fichiers liés aux tags.
- CLI ou GUI.

Points positifs	Points négatifs
Simples	Dépendance à une BDD externe
Rapides et efficaces	Modification et accès uniquement par l'app (Tagsistant et TagSpaces)
Open source	Non écrits en Rust



macOS

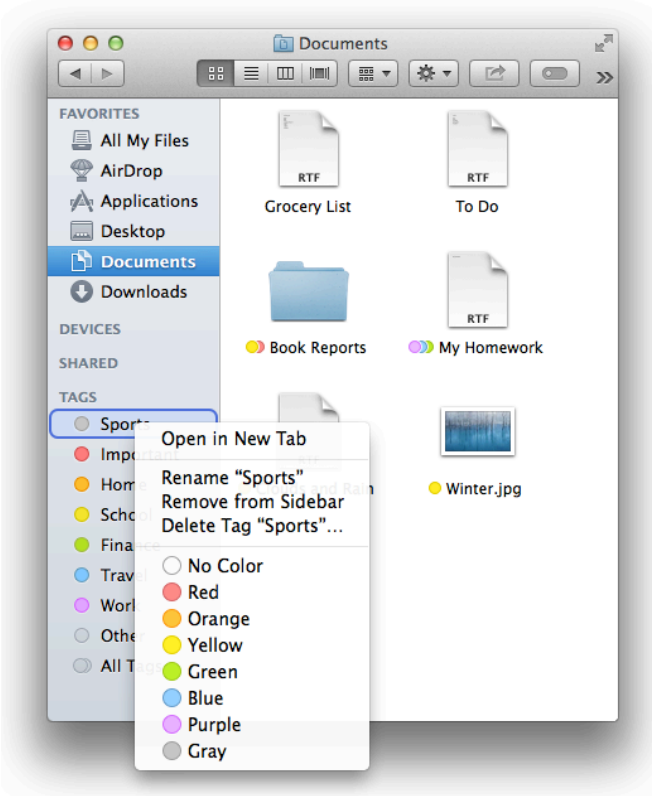
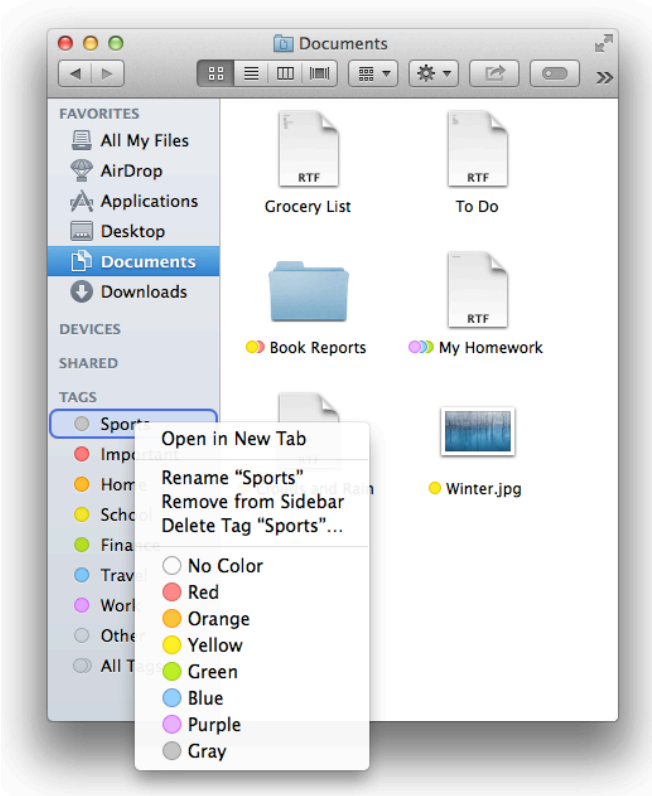


Figure – Gestion d'un tag dans le Finder [1]

macOS



Points positifs	Points négatifs
Système de tags intégré à l'explorateur de fichiers	Code propriétaire
Stocke les tags dans les attributs des fichiers	Seulement pour macOS
Performant	Non écrit en Rust

Figure – Gestion d'un tag dans le Finder [1]

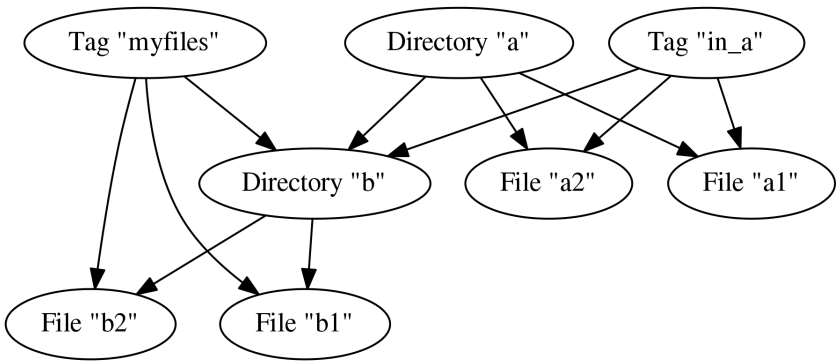
Gestion des tags

- Stockage des tags dans les attributs étendus (XATTR).

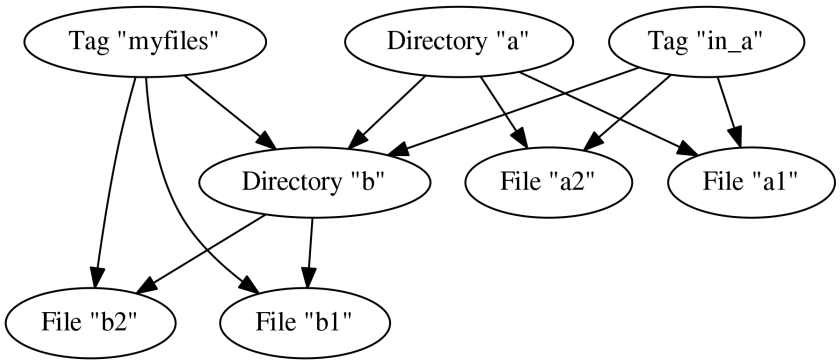
Gestion des tags

- Stockage des tags dans les attributs étendus (XATTR).
- Outil dédié plutôt que reprendre les commandes existantes => Confort d'utilisation.

Indexation des fichiers et des tags



Indexation des fichiers et des tags



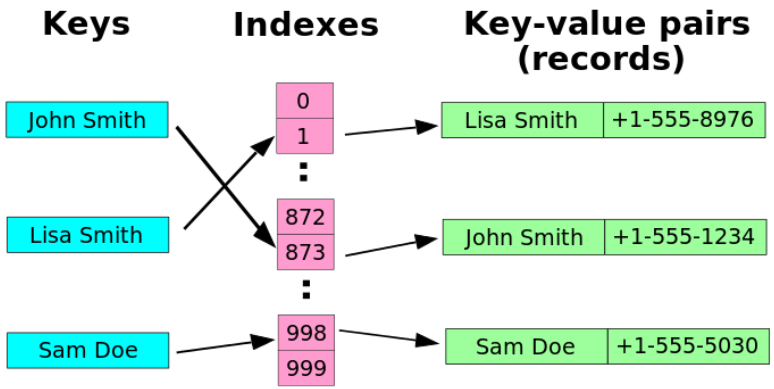


Figure – Un annuaire représenté comme une table de hachage - [2]

Surveillance du système de fichiers

Mise à jour du graphe lors des événements suivants :

Surveillance du système de fichiers

Mise à jour du graphe lors des événements suivants :

- Changement sur les tags.

Surveillance du système de fichiers

Mise à jour du graphe lors des événements suivants :

- Changement sur les tags.
- Création de fichiers/répertoires.

Surveillance du système de fichiers

Mise à jour du graphe lors des événements suivants :

- Changement sur les tags.
- Création de fichiers/répertoires.
- Suppression de fichiers/répertoires.

Surveillance du système de fichiers

Mise à jour du graphe lors des événements suivants :

- Changement sur les tags.
- Création de fichiers/répertoires.
- Suppression de fichiers/répertoires.
- Déplacement/renommage de fichiers/répertoires.

Requêtes de tags et fichiers

Requêtes de tags et fichiers

- Lister les fichiers et répertoires associés à des tags (expressions logiques).

Requêtes de tags et fichiers

- Lister les fichiers et répertoires associés à des tags (expressions logiques).
- Lister les tags existants.

Requêtes de tags et fichiers

- Lister les fichiers et répertoires associés à des tags (expressions logiques).
- Lister les tags existants.
- Renommer un tag.

Rust
Généralités (1)

- Langage moderne, performant, fiable, compilé, et fortement typé.

Rust
Généralités (1)

- Langage moderne, performant, fiable, compilé, et fortement typé.
- Disponible sur Linux, Windows et macOS.

Rust
Généralités (1)

- Langage moderne, performant, fiable, compilé, et fortement typé.
- Disponible sur Linux, Windows et macOS.
- Cargo : outil de compilation et d'exécution et gestionnaire de paquets intégré à Rust.

Rust
Généralités (1)

- Langage moderne, performant, fiable, compilé, et fortement typé.
- Disponible sur Linux, Windows et macOS.
- Cargo : outil de compilation et d'exécution et gestionnaire de paquets intégré à Rust.
- Structures, collections, généricité, énumérations et *pattern matching*.

Rust
Généralités (1)

- Langage moderne, performant, fiable, compilé, et fortement typé.
- Disponible sur Linux, Windows et macOS.
- Cargo : outil de compilation et d'exécution et gestionnaire de paquets intégré à Rust.
- Structures, collections, généricité, énumérations et *pattern matching*.
- Gestion des erreurs.

Rust
Généralités (1)

- Langage moderne, performant, fiable, compilé, et fortement typé.
- Disponible sur Linux, Windows et macOS.
- Cargo : outil de compilation et d'exécution et gestionnaire de paquets intégré à Rust.
- Structures, collections, généricité, énumérations et *pattern matching*.
- Gestion des erreurs.

```
1 enum Result<T, E> { Ok(T), Err(E), }
2 match value {
3   Ok(data) => println!("Value : {}", data),
4   Err(error) => panic!("Error : {}", error)
5 }
```

Rust
Généralités (2)

- Tests.

Rust
Généralités (2)

- Tests.
- Unsafe Rust.

Rust
Généralités (2)

- Tests.
- Unsafe Rust.
- Concurrence et Threads.

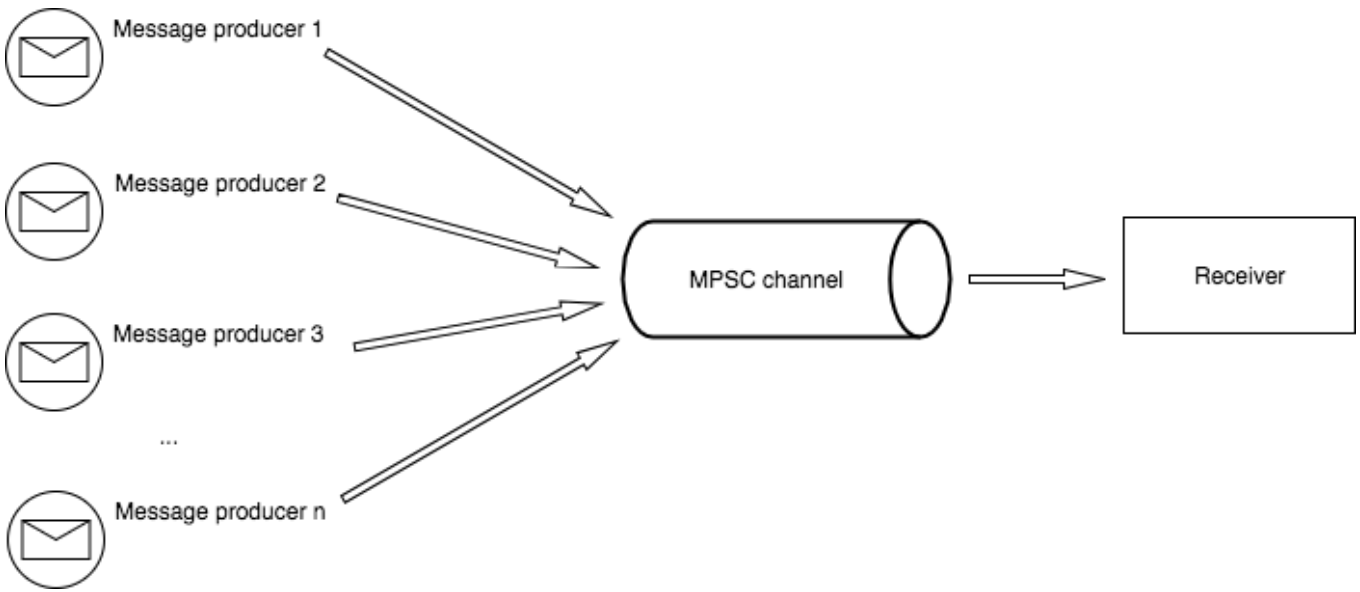


Figure – Canal de communication entre threads - [3]

Rust
Ownership

Rust
Ownership

- Chaque variable est dite le "possesseur" (*owner*) d'une valeur.

Rust
Ownership

- Chaque variable est dite le "possesseur" (*owner*) d'une valeur.
- Il ne peut y avoir qu'un seul *owner* pour une valeur.

Rust
Ownership

- Chaque variable est dite le "possesseur" (*owner*) d'une valeur.
- Il ne peut y avoir qu'un seul *owner* pour une valeur.
- Lorsque l'*owner* est détruit ou change de portée, la valeur est détruite.

Rust
Ownership

- Chaque variable est dite le "possesseur" (*owner*) d'une valeur.
- Il ne peut y avoir qu'un seul *owner* pour une valeur.
- Lorsque l'*owner* est détruit ou change de portée, la valeur est détruite.

```
1 let a = 10;
2 let b = a;
3 let mut my_vec = vec![3, 2, 1];
4 let other_vec = my_vec;
5 my_vec.push(42); // Erreur, la valeur a été déplacée
```

Rust
Borrowing

Rust
Borrowing

1. À tout moment, il ne peut exister qu'une seule référence mutable ou plusieurs références immutables, mais pas les deux en même temps.

Rust
Borrowing

1. À tout moment, il ne peut exister qu'une seule référence mutable ou plusieurs références immutables, mais pas les deux en même temps.
2. Les références doivent toujours être valides.

Rust
Borrowing

1. À tout moment, il ne peut exister qu'une seule référence mutable ou plusieurs références immutables, mais pas les deux en même temps.
2. Les références doivent toujours être valides.

```
1  fn main() {
2      let mut my_vec = vec![3, 2, 1];
3      ref_immutable(&my_vec);
4      ref_mutable(&mut my_vec);
5  }
6  fn ref_immutable(v : &Vec<i32>) { println!("{:?}", v); }
7  fn ref_mutable(v : &mut Vec<i32>) { v.push(42); }
```

Attributs étendus (XATTR)

- Métadonnées sous forme de paire `espace.nom:valeur`.

Attributs étendus (XATTR)

- Métadonnées sous forme de paire `espace.nom:valeur`.
- Nom = chaine de caractères, valeur = chaine de caractères ou données binaires.

Attributs étendus (XATTR)

- Métadonnées sous forme de paire `espace.nom:valeur`.
- Nom = chaine de caractères, valeur = chaine de caractères ou données binaires.
- Existent sous ext2-3-4, XFS, Btrfs, UFS1-2, NTFS, HFS+, ZFS.

Inotify

- API de notifications d'événements sur le système de fichiers.

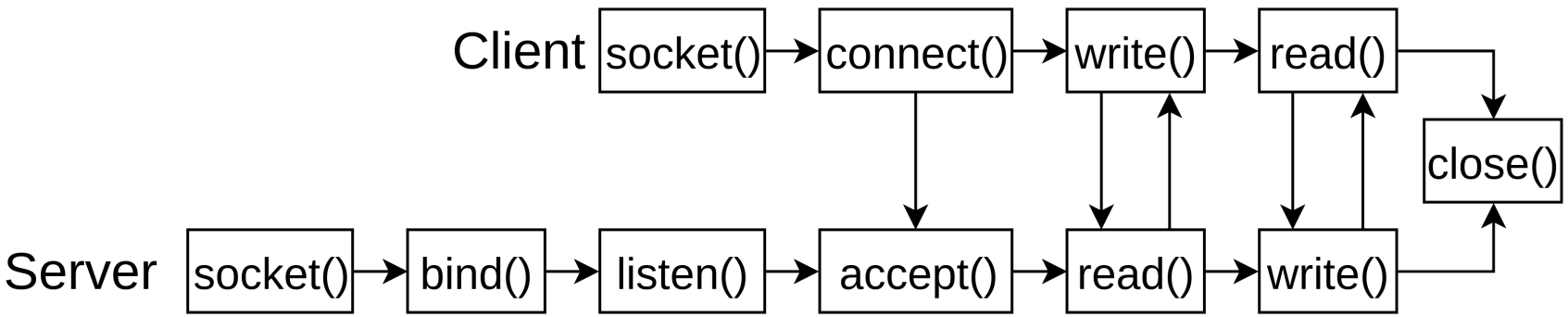
Inotify

- API de notifications d'événements sur le système de fichiers.
- Trois appels système : initialisation, ajout de surveillance sur un chemin de fichiers donné et suppression de cette surveillance.

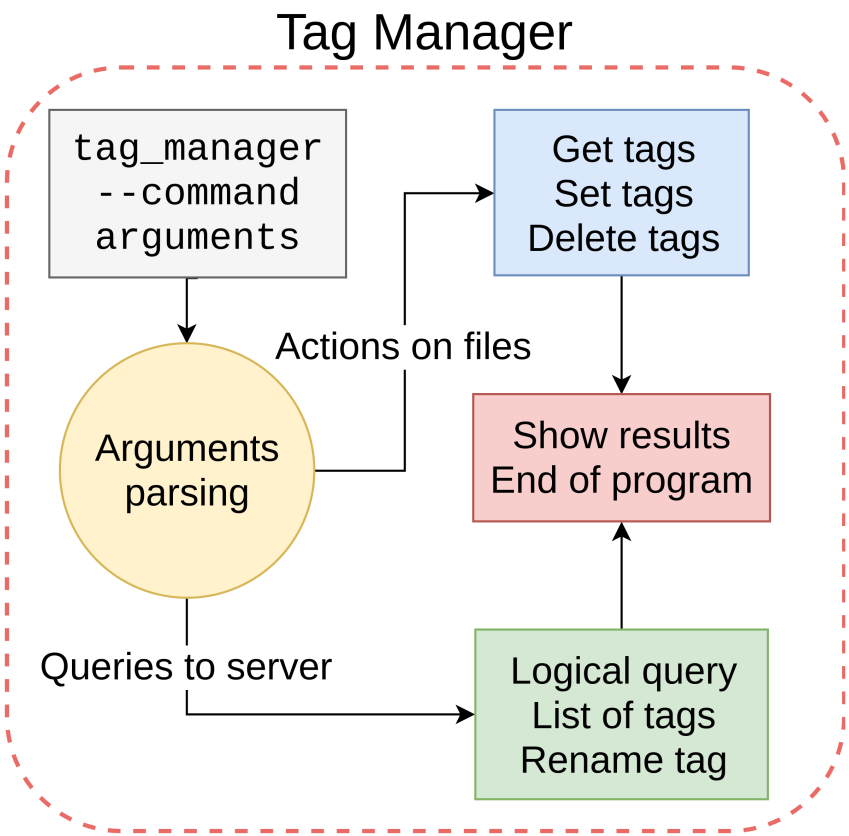
Inotify

- API de notifications d'événements sur le système de fichiers.
- Trois appels système : initialisation, ajout de surveillance sur un chemin de fichiers donné et suppression de cette surveillance.
- Lecture d'un événement avec read().

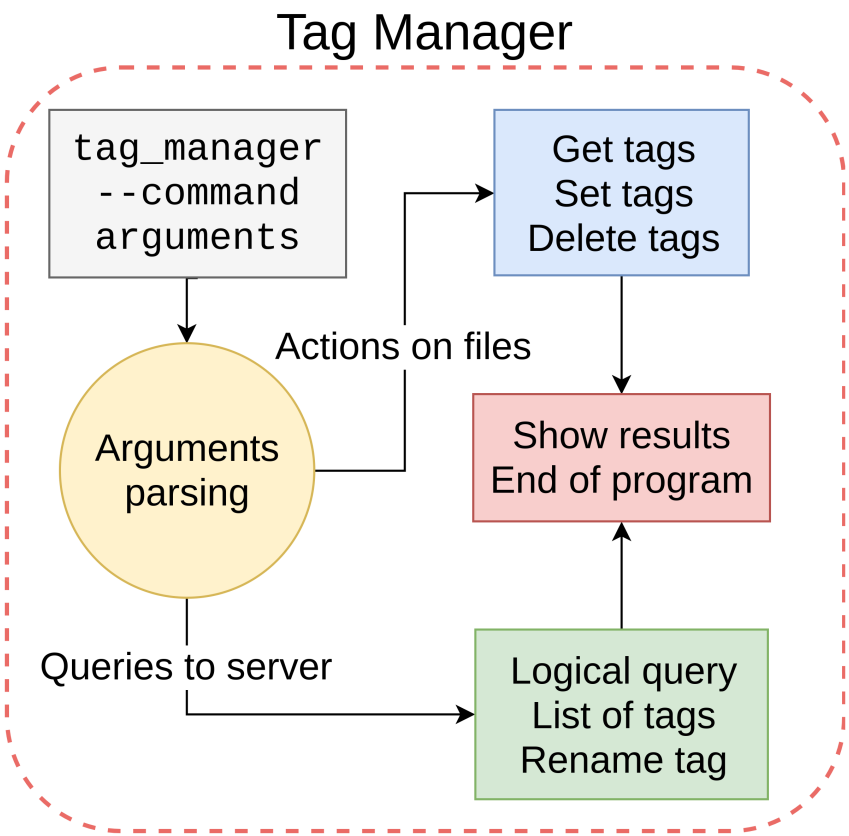
Sockets


Client
socket()
connect()
write()
read()
close()
Server
socket()
bind()
listen()
accept()
read()
write()

Tag Manager

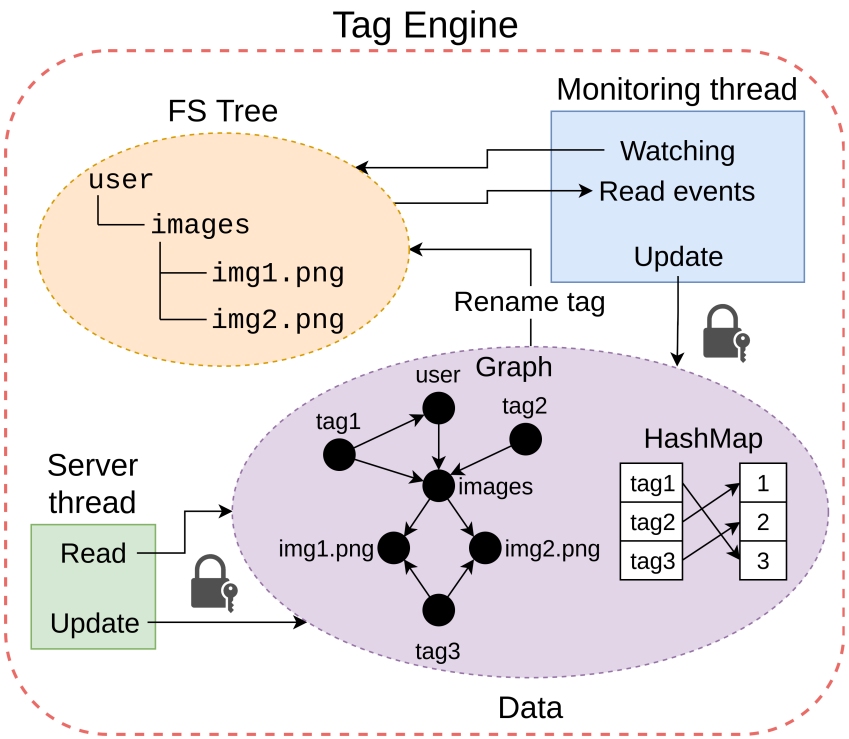

Tag Manager
tag_manager
--command
arguments
Get tags
Set tags
Delete tags
Actions on files
Arguments
parsing
Show results
End of program
Queries to server
Logical query
List of tags
Rename tag

Tag Manager

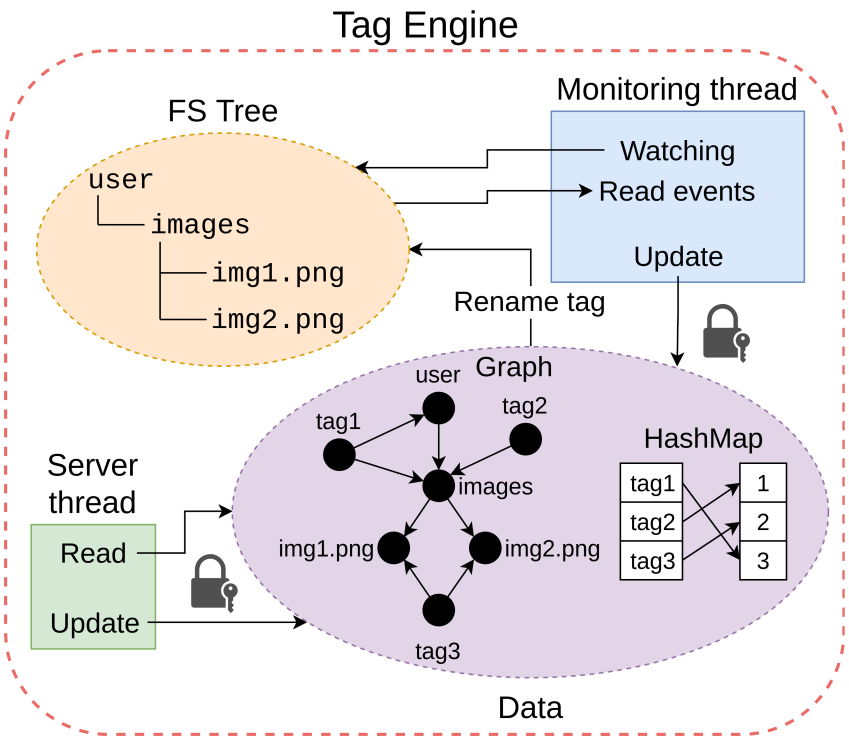


- CLI.
- Gestion des tags.
- Requêtes sur les tags et fichiers.
- Manipule les XATTR des fichiers.
- Programme "client".

Tag Engine

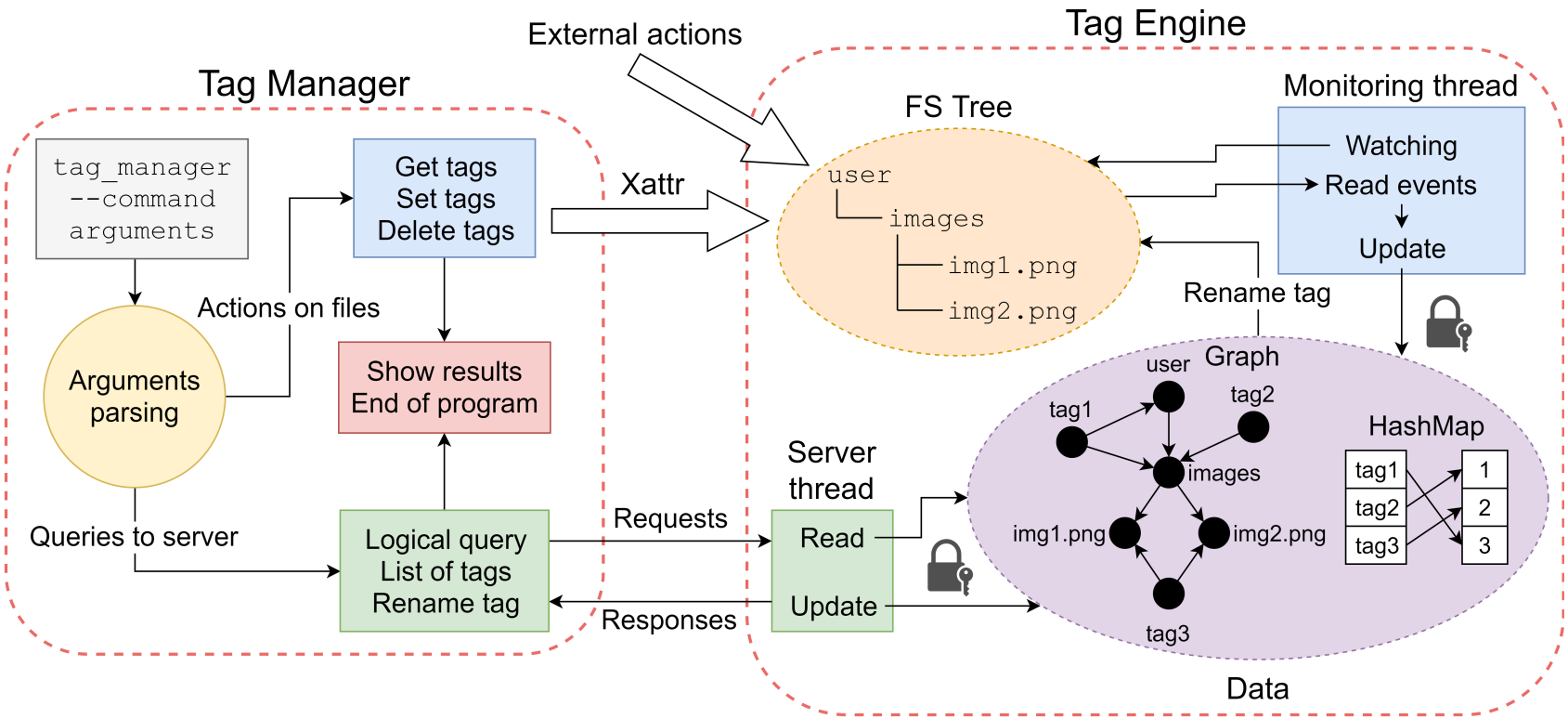

Tag Engine
FS Tree
user
images
img1.png
img2.png
Monitoring thread
Watching
Read events
Update
Rename tag
Graph
user
tag1
tag2
images
img1.png
img2.png
tag3
HashMap
tag1
tag2
tag3
1
2
3
Server thread
Read
Update
Data

Tag Engine

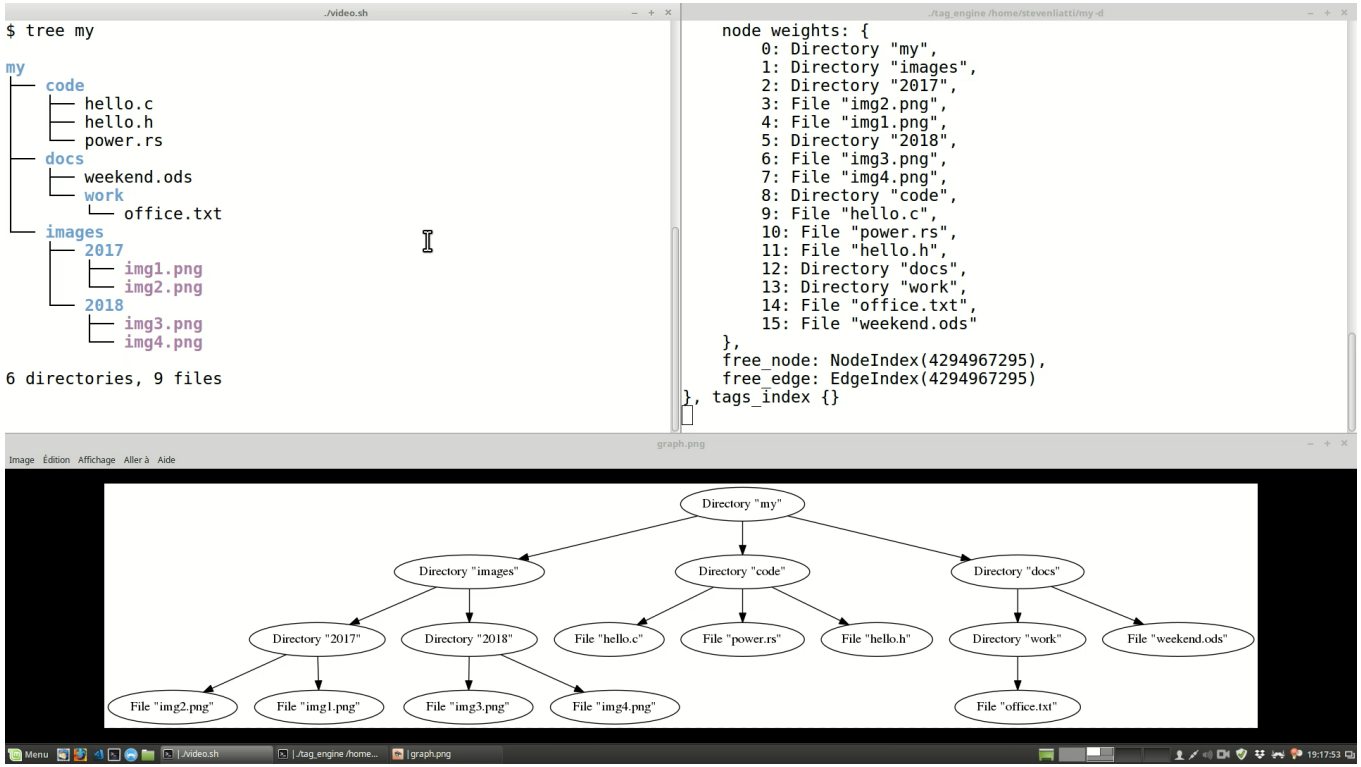


- Surveille l'arborescence des fichiers.
- Écoute sur une socket les requêtes provenant de Tag Manager.
- Maintient la relation entre tags, fichiers et répertoires à l'aide d'un graphe orienté et d'une hashmap.
- Les changements sur le FS sont répercutés sur le graphe.
- Multithread.

TagFS


External actions
Tag Engine
Tag Manager
FS Tree
Monitoring thread
tag_manager
--command
arguments
Get tags
Set tags
Delete tags
Xattr
user
images
img1.png
img2.png
Watching
Read events
Update
Actions on files
Rename tag
Arguments
parsing
Show results
End of program
user
Graph
tag1
tag2
images
HashMap
Server
thread
Queries to server
Logical query
List of tags
Rename tag
Requests
Read
Update
Responses
img1.png
img2.png
tag3
tag1
tag2
tag3
1
2
3
Data

Démo


./video.sh
$ tree my
my
├── code
│ ├── hello.c
│ ├── hello.h
│ └── power.rs
├── docs
│ ├── weekend.ods
│ └── work
│ └── office.txt
└── images
├── 2017
│ ├── img1.png
│ └── img2.png
└── 2018
├── img3.png
└── img4.png
6 directories, 9 files
./tag_engine /home/stevenliatti/my -d
node weights: {
0: Directory "my",
1: Directory "images",
2: Directory "2017",
3: File "img2.png",
4: File "img1.png",
5: Directory "2018",
6: File "img3.png",
7: File "img4.png",
8: Directory "code",
9: File "hello.c",
10: File "power.rs",
11: File "hello.h",
12: Directory "docs",
13: Directory "work",
14: File "office.txt",
15: File "weekend.ods"
},
free_node: NodeIndex(4294967295),
free_edge: EdgeIndex(4294967295)
}, tags_index {}
graph.png
Image Édition Affichage Aller à Aide
Directory "my"
Directory "images"
Directory "code"
Directory "docs"
Directory "2017"
Directory "2018"
File "hello.c"
File "power.rs"
File "hello.h"
Directory "work"
File "weekend.ods"
File "img2.png"
File "img1.png"
File "img3.png"
File "img4.png"
File "office.txt"

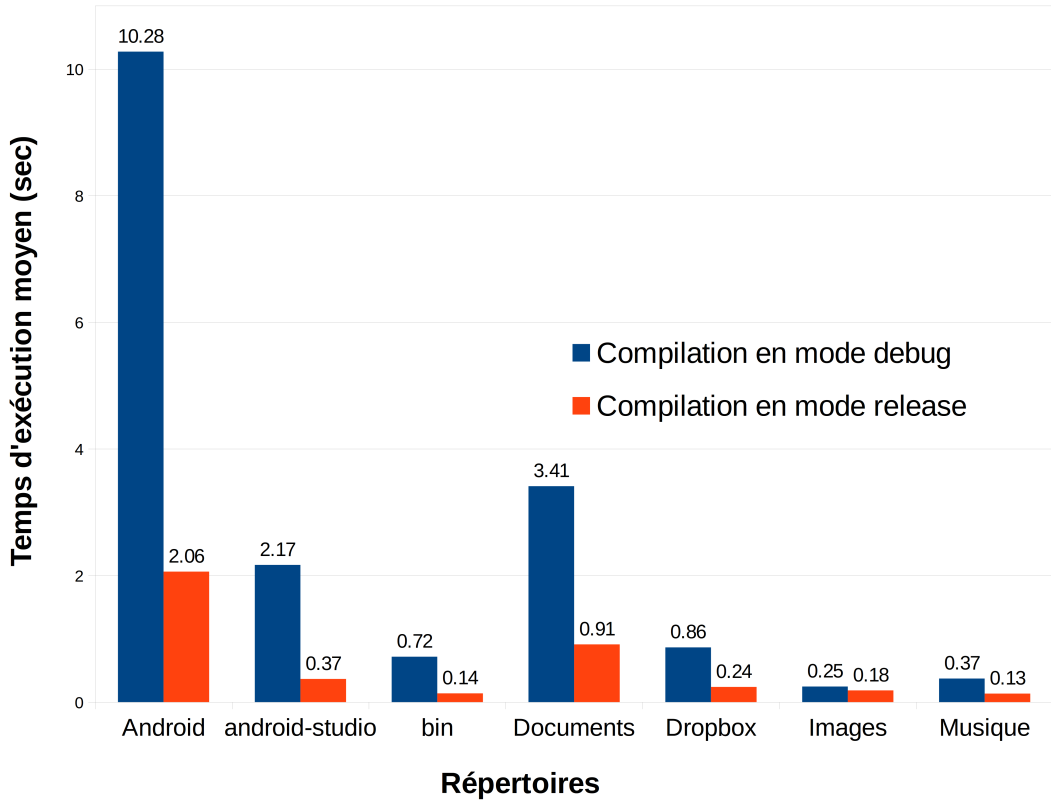
Vidéo

Mesures de performances

Répertoire	Sous-répertoires	Fichiers
Android	15'172	112'046
android-studio	3'331	13'287
bin	553	9'306
Documents	15'442	64'486
Dropbox	2'377	8'659
Images	5	863
Musique	135	1'352

Mesures de performances

Répertoire	Sous-répertoires	Fichiers
Android	15'172	112'046
android-studio	3'331	13'287
bin	553	9'306
Documents	15'442	64'486
Dropbox	2'377	8'659
Images	5	863
Musique	135	1'352



Rust vs C

Avantages par rapport à C	**Inconvénients par rapport à C**
Garanties sécurité mémoire	Courbe d'apprentissage plus longue
Détection des erreurs à la compilation	Contraintes du langage parfois handicapantes
Compilateur verbeux	Moins répandu
Performances égales ou très proches	Manque de soutien global
Gestion des erreurs (NULL)	
Cargo et Crates.io	
Librairie standard	
Généricité	

Conclusion

Conclusion

- Étude du langage Rust.

Conclusion

- Étude du langage Rust.
- Conception d'un moteur de gestion de tags.

Conclusion

- Étude du langage Rust.
- Conception d'un moteur de gestion de tags.
- Diverses technologies et approches.

Remerciements

- Florent Glück
- Orestis Malaspinas
- Joël Cavat

Références I

Apple team.

Os x : Tags help you organize your files.
https://support.apple.com/en-us/HT202754, février 2015.
Consulté le 08.05.2018.

Wikipédia.

Un annuaire représenté comme une table de hachage.
https://fr.wikipedia.org/wiki/Table_de_hachage#/media/File:HASHTB08.svg, juin 2015.
Consulté le 23.06.2018.

Marcin Baraniecki.

Multithreading in rust with mpsc (multi-producer, single consumer) channels.
https://bit.ly/2AbJELg, novembre 2017.
Consulté le 18.06.2018.