Doge FS

Raed Abdennadher et Steven Liatti

Programmation avancée des systèmes - Prof. Florent Glück

Hepia ITI 3ème année

18 décembre 2017

1 Description

Ce rapport décrit le système de fichiers "Doge FS" utilisé dans notre kernel. Doge FS est basé sur File Allocation Table (FAT), un système de fichier très répandu. Une image formatée en Doge FS contient des blocs de bytes (multiple de 512). Les tous premiers bytes d'une image (situés dans le premier bloc) contiennent les informations de base du système, nous l'appelons le "super bloc". Le(s) bloc(s) suivant(s) contien(nen)t la FAT, un tableau de listes chainées d'indices de blocs. Finalement, après la FAT, les blocs suivants contiennent soit des méta-données (des "entrées") des fichiers, soit les données proprement dites des fichiers. Ce système est non hiérarchique (pas de gestion des dossiers), tous les fichiers sont à la "racine". Voici le détail des 3 structures de données mentionnées ci-dessus :

- `super_block_t` : contient les informations de base du système :
  - `magic` : (`char`) la signature, à 0x42, ou 66 en base 10
  - `version` : (`char`) la version du système
  - `label` : tableau de `char` contenant le nom du système (ici "doge_os")
  - `block_size` : (`int`) la taille d'un bloc (multiple de 512, jusqu'à 4096)
  - `fat_len` : (`int`) le nombre total de blocs indexés par la FAT (= nombre de blocs total)
  - `fat_block_nb` : (`int`) le nombre de blocs occupés par la FAT elle-même
  - `first_entry` : (`int`) l'indice du premier bloc de méta-données
- `entry_t` : représente les méta-données, c'est une liste d'entrées de fichiers :
  - `name` : tableau de `char` contenant le nom du fichier
  - `size` : la taille en bytes du fichier
  - `start` : l'indice du premier bloc de données du fichier
- FAT : tableau de `int`. Chaque case représente un bloc du système de fichier et contient :
  - -1 : indique que le bloc est libre et n'a pas de suivant
  - 0 : indique que ce bloc est le dernier d'une liste chainée de données ou de méta-données
  - entier compris entre 2 et $fs_{size}/block_{size}$: indique l'indice du bloc suivant de données ou de méta-données

Le schéma suivant représente un exemple de disposition des structures et données sur une image Doge FS :

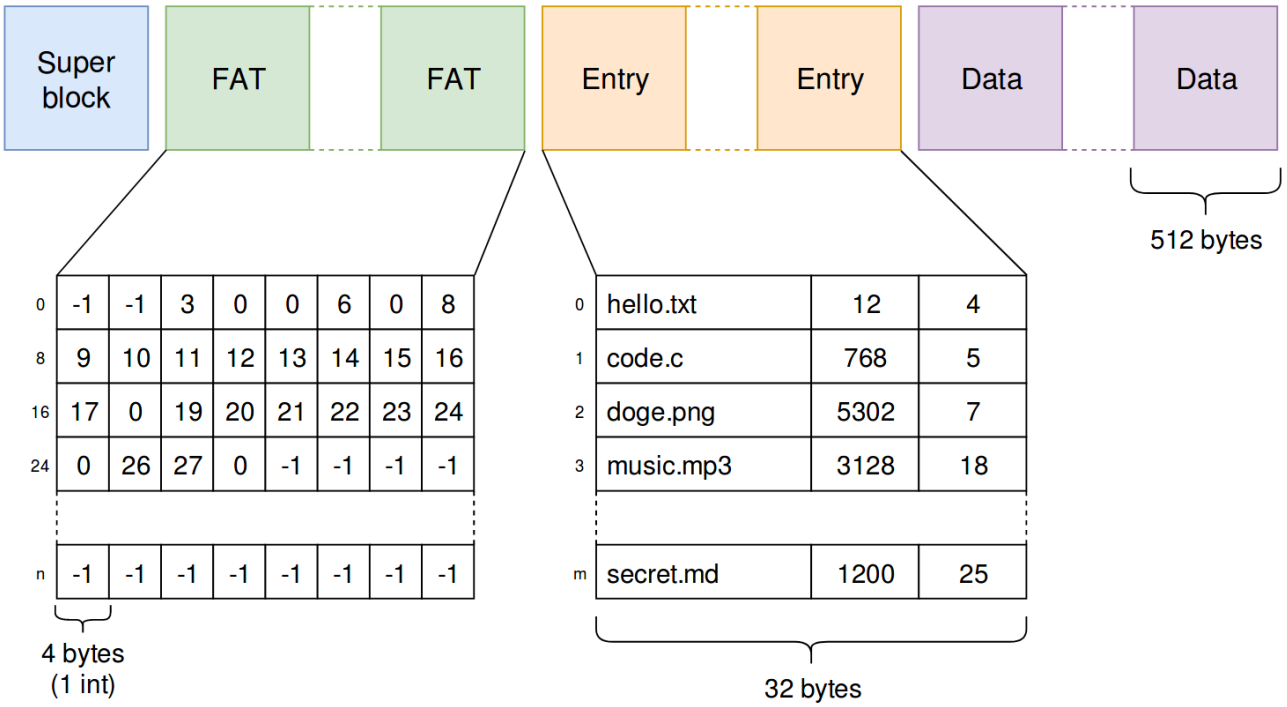

FIGURE 1 – Format du système de fichiers

Sur ce schéma, les blocs ont une taille de 512 bytes. Avec une taille maximale de label fixée à 20 caractères, le super bloc a une taille constante de 38 bytes, quelle que soit la taille de bloc choisie. Le sous-tableau de gauche représente la FAT. Elle occupe au minimum un bloc et commence au bloc 1 (ici à un offset de 512 bytes). La FAT peut contenir $block_{size}/4$ cases par bloc, ici 128. Sur le schéma, les numéros positionés à gauche de la FAT représentent les indices des cases. Les deux premières cases sont à -1, elles représentent respectivement le super bloc et la première case de la FAT elle-même. Le sous-tableau de droite représente la liste des entry (les méta-données). Avec une longueur maximale de 24 caractères pour un nom de fichier, une entrée a une taille de 32 bytes (24 bytes de nom, 4 bytes de taille et 4 bytes d'indice de premier bloc de données), par conséquent un bloc de 512 bytes peut en contenir 16 ($block_{size}/32$).
Dans cet exemple, 2 blocs sont alloués pour les méta-données, les blocs 2 et 3. Le premier fichier, "hello.txt" occupe un bloc et se trouve à l'indice 4, qui contient un zéro, indiquant qu'il s'agit de la fin de la chaine. Le deuxième fichier, "code.c" occupe 2 blocs, aux indices 5 et 6 dans la FAT. Les autres fichiers remplissent la FAT en séquence selon la même analogie. À noter qu'une fois passés les blocs de FAT, le prochain bloc sera un bloc d'entry, mais les suivants pourront contenir soit des entry, soit des données de fichiers. Cette alternance est possible grâce à la structure de liste chainée qui est en place dans la FAT. Cela apporte plus de souplesse lorsque de nombreux fichiers sont ajoutés.
Lorsque le kernel est exécuté avec une image Doge FS montée comme système de fichiers, nous chargeons en mémoire vive toute la FAT, pour obtenir directement les indices des blocs à lire/écrire sur le disque. Étant donné que l'écriture de fichiers côté kernel n'est pas requise dans ce TP, la FAT en RAM sera la seule à être consultée. Si nous devions gérer les cas d'écriture, nous pourrions imaginer un mécanisme d'écriture intermittente qui mettrait à jour la FAT sur le disque lorsque nécessaire.

2 Avantages et inconvénients

2.1 Avantages

— Implémentation aisée
— Les fichiers peuvent facilement grandir
— Pas de fragmentation externe
— Accès aléatoire rapide, grâce à la FAT chargée en RAM

2.2 Inconvénients

— Overhead important pour le stockage de la FAT (disque et RAM)
— Accès séquentiel potentiellement lent, bien que limité dans notre cas