Démo #10

Systèmes de fichiers

Qu’y-a-t’il dans un fichier?

- Données : série de bytes
- Métadonnées

Opérations

- Créer
- Ouvrir
- Fermer
- Écrire
- Supprimer
- Changer taille

Types d’accès

- Accès séquentiel : read, write, fseek
  - Un pointeur à la position dans le fichier est maintenu

- Accès direct : mmap, munmap
  - Une partie de la mémoire correspond maintenant au fichier, opérer dessus correspond à opérer sur le fichier
  - Synchronisé p/e périodiquement mais certainement à close

Fichiers dans le contexte d’un OS multi-processus

- Quel problèmes pouvez-vous anticiper?

Répertoires en graphe

Un répertoire (dossier) est un fichier qui établi une correspondance entre le nom de fichiers et les fichiers eux-mêmes

- Généralement une hash-table de nos jours
- Tout(*) est un fichier!

Cycles

- On peut ajouter des liens
  - Des vrais liens (hard-link), où le même fichier à plusieurs noms (aliasing)
  - Des liens symboliques (symlink) où un nom de fichier pointe à un autre
- Ça pourrait créer des cycles, donc pas de hardlink pour les dossiers
- Pas vraiment de raison de nos jours d’utiliser des hard links!

Permissions

Dans Linux, chaque fichier appartient à un utilisateur et à un groupe

- On a donc des permissions pour :
  - Le public général (tout le monde)
  - L'utilisateur associé (owner)
  - Le groupe associé

3 bit pour définir les permissions : rwx

- Par exemple : rwxr-xr-x
  - Tout le monde peut lire et exécuter, mais seulement le owner peut écrire
- Pour un dossier, x correspond à la permission d'entrer dans le dossier

Permission, p.2

Linux a quelque permissions spéciales

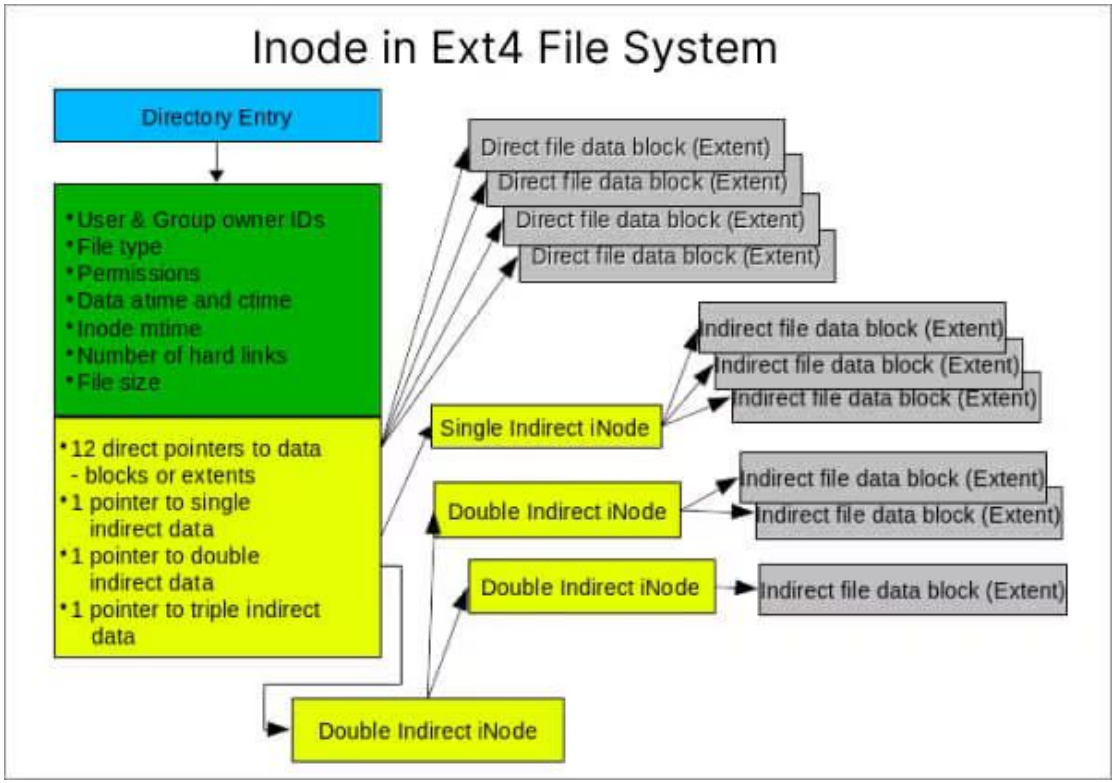
- s: SUID
  - L’éxecution du fichier se fait au nom du owner ou du groupe, dépendamment de où la permission se trouve
  - Remplace x
  - Si au niveau du groupe dans un dossier, assigne ce groupe comme groupe à tout fichier dedans par défaut
- t:
  - Seul l’owner du dossier peut supprimer des fichier à l’intérieur
  - Remplace x

FS: FAT

- FAT: File Allocation Table
- Liste chaînée d’index pour chaque bloc : chaque entrée dans la table contient l’index dans la table pour l’entrée suivante
- On stocke ensuite le dossier root dans une section spéciale
- Pas génial, car une entrée pour chaque bloc requiert pas mal d’entrées

FS: ext4

- Structuré en iNode (fichier, dossier, inodes indirectes)
- Chaque fichier peut pointer vers des data block directement, où vers un arbre de inode puis data block
- Beaucoup plus efficace!



Résilience

- Comment survivre à une perte de courant? Corruption?

Journaling, COW, etc…

- Journal : on fait un record des opérations qu'on va/est entrain de faire, pour pouvoir les reprendre ou annuler
- COW : On n'écrit pas immédiatement, mais on fait plutôt une copie
  - Moins cher car pas besoin d'effacer, et on peut restaurer l'original
- Duplication et parité
  - Soit seulement métadonnées, car plus dangereuses, soit tout
- Duplication à travers plusieurs devices physiques (RAID, ZFS)
- Snapshot
  - On garde les métadonnées pour un état du FS, les données restent après le COW

Mount

- Le FS par défaut se met à la racine
- On peut adjoindre des FS en mettant leur racine comme dossier (mountpoint)

Partage de systèmes de fichier

Un ordinateur qui contient le système de fichier peut exposer celui-ci en exposant les méthodes d’accès au fichier comme API

- NFS
- CIFS/Samba
- ZFS
  - ZFS est un FS local et partagé

Exercices

- Quel est l'impact sur la performance d'utiliser un système de fichier avec indirects (ext4) vs avec liste chaînée de blocs?
- Comment est-ce que le type de stockage affecterait la conception d'un système de fichiers?
- Quelle partie du FS devrait le plus possible rester dans la cache? Quel problème est-ce que ça pourrait toutefois engendrer?