PNL - Projet

`ouiche_fs` – le système de fichiers le plus classe du monde

Redha Gouicem, Julien Sopena

Dans ce projet, vous implémenterez de nouvelles fonctionnalités dans un système de fichiers existant, `ouiche_fs`, sous la forme d'un module compatible avec la version 4.19.3 de Linux. N'hésitez pas à utiliser un site de référencement croisé du code du noyau Linux pour vous aider (par exemple https://elixir.bootlin.com/linux/v4.19.3/source). Il est également fortement recommandé de lire la documentation du fichier `Documentation/filesystems/vfs.txt` présent dans les sources du noyau pour mieux comprendre l'implémentation d'un système de fichiers.

Rappel : Lire attentivement l'énoncé en entier !

Dans un premier temps, téléchargez les sources de `ouiche_fs` sur GitHub à l'adresse suivante : https://github.com/rgouicem/ouichefs.

En plus des sources, vous trouverez des explications sur la conception de ce système de fichiers, ainsi que les relations entre les structures de données du noyau entrant en jeu dans `ouiche_fs`. N'hésitez pas à lire les sources (amplement commentées) et à expérimenter `ouiche_fs` sur votre machine virtuelle afin d'en comprendre le fonctionnement.

1 Déduplication de blocs

Plusieurs techniques existent pour économiser de l'espace de stockage. Parmi elles, la déduplication de blocs consiste à détecter les blocs identiques sur la partition pour n'en conserver qu'un seul exemplaire.

Dans un premier temps, vous devrez implémenter la déduplication de blocs en considérant que vos fichiers ne seront pas modifiés. La déduplication peut être effectuée **en ligne** (lors de l'écriture d'un bloc, on cherche s'il existe déjà sur la partition pour ne pas le créer de nouveau dans ce cas) ou **hors ligne** (toute la partition est parcourue pour trouver les doublons à posteriori, lorsque la partition n'est plus utilisée). Dans Linux, la déduplication en ligne est difficile car la page cache et la couche d'accès aux périphériques de type bloc peuvent entrer en conflit avec cette manipulation. Il est donc *conseillé* d'effectuer la déduplication hors ligne, lors du démontage de la partition par exemple.

Vous devrez aussi tenir compte du problème suivant : lorsqu'un bloc n'est pas complètement utilisé pour stocker des données, les bits restants ont un contenu aléatoire (le contenu précédent du bloc avant sa réallocation par exemple). Deux blocs avec un contenu utile identique peuvent donc être différents.

Pour tester le bon fonctionnement de votre implémentation, vous pouvez par exemple créer plusieurs versions du même fichier, et vérifier qu'après déduplication, tous les fichiers partagent leurs blocs. L'espace disponible sur votre partition devrait également augmenter (utilisez `df -h`). Pensez à imaginer d'autres tests pour valider votre algorithme de déduplication.

2 Copie sur écriture

La copie sur écriture (*copy-on-write* ou *cow*) consiste à dupliquer une donnée partagée lorsqu'un des propriétaire de cette donnée modifie celle-ci. Dans le cas de `ouiche_fs`, après déduplication, certains blocs seront partagés par plusieurs fichiers. Si un de ces fichiers est modifié, seul ce dernier doit pouvoir observer cette modification. Les autres fichiers doivent conserver la version originale des blocs modifiés. Pour savoir si une telle duplication est nécessaire, il faut savoir combien de fichiers partagent un bloc. N'hésitez pas à modifier le programme `mkfs` fourni si nécessaire.

Implémentez cette fonctionnalité et testez la en modifiant des fichiers partageant des blocs dédupliqués.

3 Bonus : Correction de bugs

Si vous décelez des bugs dans `ouiche_fs`, n'hésitez pas à les signaler et à les corriger (en échange de quelques points bonus bien sûr...).

4 Soumission

4.1 Procédure

Vous devrez soumettre votre travail sur la plateforme Moodle sous la forme d'une archive au format `tar.gz`. Cette archive devra contenir les sources modifiées de `ouiche_fs` sans résidus de compilation, ainsi qu'un court rapport au format `pdf`. Ce rapport contiendra les explications de vos choix de conception (notamment les algorithmes et les structures de données utilisés), ainsi que l'état d'avancement de votre projet. Si vous rencontrez des problèmes pour lesquels vous ne trouvez pas de solution, expliquez ces problèmes, leurs origines ainsi que des pistes de résolution. Ce projet est à réaliser en **binôme**. Ces binômes seront à constituer sur la plateforme Moodle avant le **25 avril 2019**. Le projet est à soumettre au plus tard le **lundi 3 juin 2019 à 9h00**.

4.2 Évaluation

Vous serez bien entendu évalués sur le fonctionnement de votre projet, mais également sur la qualité et la lisibilité de votre code (pensez à utiliser le script `checkpatch.pl` notamment) et de votre rapport.

5 Annexe

5.1 Buffer cache (include/linux/buffer_head.h)

`struct buffer_head *sb_bread(struct super_block *sb, sector_t block bno);` : renvoie un pointeur sur le buffer en cache du bloc `bno` de la partition `sb`. La taille du buffer lu correspond à `sb->s_blocksize`, et les données de ce buffer sont accessible via le champ `b_data` de la `struct buffer_head` renvoyée.

`void mark_buffer_dirty(struct buffer_head *bh);` : signale au buffer cache que le buffer `bh` a été modifié et devra être recopié sur le disque.

`int sync_dirty_buffer(struct buffer_head *bh);` : force l'écriture du buffer `bh` sur le disque. Renvoie 0 en cas de succès.

`void brelse(struct buffer_head *bh);` : relâche la référence sur le buffer `bh`.

5.2 Inode (include/linux/fs.h)

`void mark_inode_dirty(struct inode *inode);` : signale une modification de l'inode. Lorsque celle-ci sera libérée, la fonction `destroy_inode()` sera appelée sur cette inode.

5.3 Page cache

`void *kmap(struct page *page);` : associe une adresse virtuelle de l'espace d'adressage du noyau à l'adresse physique de la page passée en paramètre et retourne cette adresse virtuelle. Un appel à `kunmap()` est nécessaire lorsque l'association n'est plus utilisée.

`void kunmap(struct page *page);` : dissocie une adresse virtuelle de l'espace d'adressage du noyau de l'adresse physique de la page passée en paramètre.

`struct page *find_get_page(struct address_space *mapping, pgoff_t offset);` : retourne la `offset`-ème page associée à `mapping` et incrémente le compteur de référence de cette page.

`void put_page(struct page *page);` : rend la référence à `page` et décrémente son compteur de références.