Projet PNL 2019/2020

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

Auteurs :
Pierre-Loup Gosse
Salim Toufik Nedjam
Youcef Mcirdi

Encadrants :
Redha Gouicem
Rémi Oudin
Julien Sopena

8 Juin 2020

Table des matières

1 Implémentation

Le mécanisme de rotation de fichier s'exécute lorsqu'une des deux conditions est vérifiée. Notamment lors de la création d'un fichier dans la fonction `ouichefs_create`, mais aussi lors de l'écriture d'un fichier dans la fonction `ouichefs_write_begin`. C'est la fonction `ouichefs_fblocks` sera appelée avec comme paramètre l'inode du dossier source où il faut libérer des blocs.

Notre conception reste assez simple mais assure un bon fonctionnement. La fonction `ouichefs_fblocks` à pour rôle de trouver une victime à partir de l'inode du dossier source et de la supprimer. Pour ce faire nous avons découpé notre mécanisme en plusieurs parties afin de le rendre plus souple et générique.

1.1 Mécanisme de rotation

1.1.1 Structure

La suppression d'une inode victime se fera par l'utilisation de fonction déjà existante, demandant de connaître l'inode du dossier parent de la victime. L'ajout d'une structure de parenté a été ajouté :

```
struct ouichefs_inode_kinship {
    struct inode *parent;
    struct inode *inode;
};
```

1.1.2 Stratégie

Pour permettre à notre mécanisme d'être modulaire nous avons implémenté une fonction de comparaison, dite de stratégie, qui permet de comparer deux inodes en fonction de leurs champs.

```
int (*ouichefs_fblocks_strategy) (struct inode *a,
    struct inode *b) = ouichefs_fblocks_strategy_mtime;

EXPORT_SYMBOL_GPL(ouichefs_fblocks_strategy);
```

La stratégie appliquée par défaut est `ouichefs_fblocks_strategy_mtime` qui compare les deux inodes en fonction de leur date de modification. Si la

fonction retourne un résultat supérieur à 0 alors l'inode passée en second paramètre sortira gagnante de la comparaison, et sera ici considéré comme la victime.

1.1.3 Action

Une fonction dite d'action se chargera pour une inode donnée de déterminer si elle est la nouvelle victime après une comparaison avec la stratégie mis en place.

```
void ouichefs_fblocks_action(struct inode *dir,
    struct inode *inode,
    void **data)
{
    struct ouichefs_inode_kinship **victim;
    int ret = 0;

    victim = (struct ouichefs_inode_kinship **) data;

    if ((*victim)->inode == NULL)
        ret = 1;
    else if (ouichefs_fblocks_strategy != NULL)
        ret = ouichefs_fblocks_strategy((*victim)->inode, inode);

    if (ret > 0) {
        (*victim)->parent = dir;
        (*victim)->inode = inode;
    }
}
```

1.1.4 Itération

Une fonction générique d'itération permet d'itérer sur tous les inodes d'un inode source. Cette fonction prend en paramètre la fonction `action` à appliquer à chaque inode représentant un fichier, ainsi qu'une variable `data` qui sera fournie à la fonction action et qui représente ici la victime.

```
void ouichefs_iterate(struct inode *dir,
        void (*action)(struct inode *dir,
```

```
                struct inode *inode, void **data),
    void **data);
```

1.1.5 Libération de blocs

C'est la fonction `ouichefs_fblocks` qui se chargera de lancer l'itération sur les inodes du dossier source. Elle appellera `ouichefs_iterate` avec comme paramètre la fonction `ouichefs_fblocks_action` et un pointeur vers la structure de la victime.

Une fois l'itération effectuée la victime contiendra l'inode que nous devons supprimer afin de libérer des blocs. Cependant l'inode peut avoir un `dentry` qui lui est lié :

```
dentry = d_find_any_alias(victim->inode);
```

Si `dentry` ne vaut pas `NULL` alors sa suppression se fait en utilisant `vfs_unlink`, la même fonction utilisée lors d'un appel système `unlink`. Sinon nous supprimons directement l'inode avec `ouichefs_remove`, une version de `ouichefs_unlink` qui ne prend pas en paramètre de `dentry` pour l'inode.

1.2 Politique de suppression avec module

La variable `ouichefs_fblocks_strategy` étant exportée la modification de la politique de suppression est un jeu d'enfant. L'insertion du module `ouichefs_strategy_changer` permet de modifier la stratégie avec une comparaison sur le taille des fichiers, il suffit de remplacer le pointeur de fonction `int (*ouichefs_fblocks_strategy)(struct inode *a, struct inode *b)` par la nouvelle fonction qui permet de comparer deux inode.

1.3 Interaction user / fs

Nous avons décidé d'implémenter l'interaction user / fs avec le système d'ioctl.

2 Fonctionnalités implémentées

— Libération lorsque tous les inode d'un répertoire ont été utilisés.

— La suppression doit se faire si le i_count est supérieur à 1 car on a remarqué que si le fichier est caché dans les d_entry sont i_count augumente de 1 alors qu'il n'est pas nécessairement utilisée, par exemple dans le cas de la creation d'un fichier.
— Changement de politique de suppression de fichiers.
— Déclanchement du mécanisme de libération depuis le userspace.

3 Fonctionnalités à travailler

Lorsqu'il ne reste que x% de blocs libres les blocs sont bien libérés mais les fichiers ont l'air corrompus. Ceci est peut-être dû à une mauvaise fonction de suppression qui ne libère pas bien les méta-données, en effet la somme des fichiers créée dépasse la taille de la partition.

Cependant, la libération des blocs à partir du pourcentage est bien fonctionnelle car le nombre de blocs libres augmente de la taille du fichier supprimé dès que l'on atteint le seuil fixé.

L'ajout de mutex lors de la recherche des fichiers mais aussi lors du changement de pointeur de la fonction stratégie.

4 Fonctionnalités non implémentées

Toutes les fonctionnalités demandées ont été implémenté.