> Charly Oudy 3414931
>
> Hugo Lopes 3671195
>
> Pierre-Louis Boeshertz 3670239

PNL - Projet 2019/2020

ouiche_fs - le système de fichiers le plus classe du monde

Après compilation, deux modules sont créé :

- ouichefs.ko : fonctionnant sur linux 4.19.3 original
- ouichefs_sys.ko : incluant le syscall de la question 2.2, mais nécessitant de patcher le kernel.

---

1. Rotation du système de fichier

Principes

Stratégie générale

L'idée de base est de placer dans la fonction `ouichefs_create()` une vérification de l'espace disque libre puis du nombre de fichiers dans le répertoire courant, afin de déclencher le nettoyage, soit du plus gros/plus vieux de la partition et/ou du répertoire courant.

Détection du % de blocs libres

Nous avons créé une macro dans `ouichesfs.h` qui permet à l'utilisateur avant compilation de déterminé quel sera la limite du % de bloc libres.

Une fonction `check_limit()` se charge d'analyser le superblock et de renvoyer un booléen pour indiquer si la limite est dépassée ou non.

Détection d'un répertoire plein

Une fonction `is_dir_full()` se charge de parcourir le bloc du répertoire courant, et de compter les entrées dans ce bloc. Les inodes utilisées étant rangées de façon contigu dans le

bloc, on s'arrête lorsque une inode = 0. Retour booléen indiquant si le répertoire est plein (128 fichiers) ou pas.

Stratégie de suppression

Détection du plus vieux (date de dernière modification) ou du plus gros dans la partition

Nous avons créé 2 fonctions `oldest_in_partition()`, `biggest_in_partition()`, qui parcourent les inodes de toute la partition grâce au bitmap en faisant appel à la fonction `find_next_zero_bit()` nous permettant ainsi d'analyser la date de dernière modification/taille uniquement des inodes utilisées. Elle renvoie le numéro d'inode.

Après cela, soit une `structure dentry` pointant sur l'inode existe en cache, et on connait directement le répertoire à nettoyer, soit il faut rechercher dans quel répertoire est localisé le fichier à supprimer.

Localisation du fichier trouvé et de son parent

Nous avons créé la fonction `find_parent_of_ino()` qui renvoi l'inode du parent. Pour ce faire, nous parcourons de façon récursive toute l'arborescence jusqu'à trouvé l'inode recherchée. Une fois trouvée, nous savons dans quel bloc elle est, et donc l'inode de son parent.

Détection du plus vieux ou du plus gros dans le répertoire courant

Nous avons créé 2 fonction `biggest_in_dir()`, `find_oldest_in_dir()` ` qui parcourent tous le bloc de l'inode du répertoire courant, pour y trouver le plus gros ou le plus ancien. Elle retourne le numéro d'inode concernée.

Cas particulier du fichier déjà ouvert par un autre processus

Comme une inode à son compteur de référence incrémenté à chaque fois qu'une structure pointe dessus, ce compteur est à 1 juste après la création, car une structure dentry en cache pointe toujours sur l'inode. Il a fallu faire donc une vérification judicieuse du compteur de référence de l'inode en prenant en compte cette problématique en vérifiant su un dentry était présent ou non.

2.1 Politique de suppression de fichiers

Notre objectif était qu'après l'insertion d'un module, nous puissions changer la politique de suppression et passer à la suppression des plus gros fichiers, soit dans le répertoire, soit dans la partition.

Pour se faire, nous avons dû rajouter une variable global “policy” de type enum TypePolicy placée dans `policy.h`. Cette variable peut prendre 2 valeurs différentes en fonction de la politique de nettoyage après l’insertion du module: “biggest” ou “oldest” .

Afin que les fichiers `inode.c` et `policy.c` (notre module à insérer) puissent communiquer entre eux, policy est déclarée en extern. Lorsqu'on insère le module ouichefs, nous avons mis la variable policy dans un export_symbole dans le fichier `fs.c`. C'est ainsi que ouichefs peut prendre en compte la modification de policy lorsque celle-ci est modifiée par l'insertion d'un autre module.

2.2 Interaction user / fs

Afin que l'utilisateur puisse déclencher lui-même le mécanisme de libération d'espace disque, nous avons décidé d'utiliser un appel système. Comme l'appel système nécessite un patch du noyau, nous avons décidé de créer un deuxième fichier fs.c, appelé fs_sys.c, afin que vous puissiez tester notre projet sans avoir à patcher votre noyau. Le patch est disponible dans src_mod/syscall/ .

Le problème que nous avons lors de la création du syscall est que nous devons lancer clean_it() afin de déclencher le mécanisme de libération d'espace disque. Cependant, cette fonction est seulement disponible depuis le module, et donc pas depuis le répertoire du kernel; Nous avons donc du trouver un moyen de réussir à intercepter le syscall afin de lancer clean_it(). Vous verrez notre première tentative dans src_moc/syscall/syscall.c qui fonctionnait correctement mais qui nécessitait de changer le fichier .config. En effet, dans *kallsyms*, la *sys_call_table* ne s'y trouvait pas car *# CONFIG_KALLSYMS_ALL is not set*. Ne voulant pas modifier le fichier de config (qui est différent selon l'utilisateur qui va patcher son noyau), nous avons décidé de wrapper le syscall. Ceci est fait dans src_fs/fs_sys.c et nous permet d'appeler clean_it() depuis le user land.

Afin d'appliquer le patch, il suffit de lancer la commande *patch -p0 < XXX/ouichefs/src_mod/syscall/patchouichefs* depuis la racine du répertoire de votre kernel (obligatoire pour que ça marche). Le path de patchouichefs est évidemment relatif au path de votre répertoire de kernel . Le fichier testsyscall.c permet de tester le syscall depuis le kernel émulé via QEMU.

---

3 BUG DE OUICHEFS

Voir explication et script de reproduction dans le fichier tests/README.md