PNL - MU4IN402

Projet Ouichefs, saison 4 – Liens sans limites

Aymeric Agon-Rambosson, Baptiste Pires, Julien Sopena

mai 2023

Consignes à bien lire avant de vous lancer

Avant toute chose, on rappelle que chaque étape dépend de la précédente et que l'étape 5 est difficile et ne doit vraiment être faite que quand tout le reste a été **parfaitement** fait.

Vous rendrez votre travail sous la forme d'une archive contenant :

- dans un répertoire `src`, vos sources **propres** (sans résidus de compilation) avec un `Makefile` fonctionnel. Vos sources respecteront **à la lettre** le style du noyau.
- dans un fichier `README` dont un patron vous est fourni, préciser pour chaque étape si la fonctionnalité a été implémentée, si elle est fonctionnelle, et le cas échéant les erreurs que vous rencontrez. Vous présenterez de manière proprement rédigée le raisonnement derrière votre solution, et l'architecture de celle-ci.
- dans un répertoire `test`, un ensemble de scripts permettant de tester vos fonctionnalités. Vous préciserez pour chacun d'eux le prédicat qu'il se propose de tester, et si le test est concluant.

Lors de la journée de soutenance, le 31 mai, chaque groupe présentera son travail et fournira une démonstration rapide des fonctionnalités de son implémentation.

La notation se fera sur la soutenance et le rendu. Il est vivement conseillé de soigner la rédaction et la réflexion : une rédaction et une présentation convaincante, qui montrent votre bonne compréhension des concepts et des problèmes du sujet, pourront nous aider à être un peu indulgents sur une implémentation faible (dans une limite raisonnable).

`ouichefs`, le système de fichiers le plus classe du monde, est un système de fichiers simple pour Linux implémenté par le Dr. Redha Gouicem. Les sources de système de fichiers sont disponibles, et à télécharger, à l'adresse suivante : `https://github.com/rgouicem/ouichefs`.

Le but du projet est de proposer dans `ouichefs` un système de liens physiques améliorés : en plus de permettre l'utilisation des liens physiques classiques (qui ne sont actuellement pas supportés), on veut pouvoir faire des liens physiques entre des partitions `ouichefs` différentes.

`ouichefs` est livré non seulement avec le module qui implémente le système de fichiers, mais aussi avec un programme applicatif qui permet de formater le système de fichiers (voir le répertoire `mkfs`). Attention, pour certaines étapes de ce projet, vous serez amené à modifier cet utilitaire de formatage.

Afin de vous permettre d'entrer progressivement dans le projet, nous avons découpé ce dernier en quatre parties, dont la difficulté est incrémentale. Vous devez absolument les faire dans l'ordre, puisque les fonctionnalités implémentées seront nécessaires aux étapes suivantes. Les quatre parties du projet sont les suivantes :

Partie 1, Ajout de liens physiques : application des méthodes de programmation vues ce semestre ;
Partie 2, Ajout d'un UUID : lecture du code noyau et compréhension du VFS ;
Partie 3, Liens physiques inter-partitions : conception d'une nouvelle fonctionnalité pour dans le noyau
Partie 4, Migration des blocs : Partie plus complexe, contenant un bonus pour vous occuper cet été

Partie 1 : Ajout des liens physiques standard

Cette partie ne contient que des applications directes des méthodes de programmations et des API noyau que vous avez vu en cours et en TP. Elle est facilement faisable pour n'importe qui a fait les TP sérieusement.

On rappelle qu'un lien physique est un élément d'une arborescence de fichiers (un nom, ou encore une entrée d'un répertoire) qui fait référence à une inode. Quand on crée un fichier, on crée en fait une inode et un nom qui référence cette inode. Ce système d'indirection entre un nom et une inode (ce dernier seulement identifiant vraiment le fichier) permet entre autres d'avoir plusieurs noms qui correspondent au même fichier de manière transparente pour les applications (plusieurs liens physiques faisant référence à la même inode).

Chaque inode est munie d'un compteur de référence qui compte le nombre de liens physiques qui y font référence (pour cette raison, il est aussi bien nommé "compteur de liens"). Quand on utilise une commande comme `rm(1)`, on supprime en fait un lien, pas nécessairement le fichier. De fait, l'appel système utilisé par `rm(1)` est `unlink(2)`, qui porte bien son nom. Seulement si, à la suite de la suppression d'un lien, le compteur de liens tombe à 0, alors l'inode et les blocs de données correspondants doivent être libérés.

Dans `ouichefs`, il n'est pas possible de créer des liens physiques supplémentaires vers des fichiers. Autrement dit, le compteur de liens ne vaut jamais strictement plus que 1.

Étape 1 – Liens physiques standards

Dans un premier temps, vous implémenterez les liens physiques avec cette sémantique. Vous pourrez tester votre implémentation avec les commandes `ln(1)` ou `link(1)`, qui permettent de créer des liens, et `stat(1)` qui permet d'obtenir des informations sur l'inode pointée par le nom passé en paramètre. Comme dans linux, vous vous limiterez aux fichiers réguliers.

Vous lirez avec profit la page `https://www.kernel.org/doc/html/v5.10/filesystems/vfs.html` de la documentation du noyau, en particulier la partie sur les opérations d'inode.

Étape 2 – Surveillance des liens physiques

Vous implémenterez ensuite un utilitaire de débogage bien pratique pour la suite, en ajoutant un répertoire `ouichfs_part` dans le `sysfs` qui contiendra un fichier par partition `ouichefs` montée. La lecture de ces fichiers indiquera, pour chaque partition, le nombre d'inodes totales et le nombre d'inodes pointées par au moins deux liens physiques (ainsi que toutes les informations que vous jugerez utiles).

Étape 3 – Lister et supprimer des liens physiques

Dans un deuxième temps, on vous demande d'ajouter une requête `ioctl(2)` qui retourne tous les liens physiques qui pointent vers la même inode que le descripteur de fichier passé en paramètre.

Ajoutez ensuite un appel système avec le prototype suivant :

```
int unlinkall(const char *pathname);
```

Cet appel système doit supprimer tous les liens physiques qui pointent vers la même inode que le nom passé en paramètre (le nom passé en paramètre doit être supprimé lui aussi). Notez que si cet appel système termine sans erreur, le fichier sous-jacent doit avoir disparu (tous ses liens ont été supprimés). Vous devez traiter les mêmes erreurs que `unlink(2)`.

Partie 2 : Ajout d'un UUID

Cette deuxième partie requiert de lire un peu plus attentivement le code du noyau pour comprendre comment sont gérés les UUID. Il vous faudra aussi avoir bien compris le fonctionnement du VFS et comment `ouichefs` s'y intègre.

Dans son implémentation actuelle, `ouichefs` n'intègre pas **l'identifiant unique universel**, ou UUID, qui permet à Linux d'identifier de manière persistante les partitions et que vous avez déjà manipulé en TP.

Étape 4 – Identification unique des partitions `ouichefs` via UUID

Ajoutez un mécanisme permettant d'identifier chacune des partitions `ouichefs` avec un UUID. Cette identification doit bien entendu persister aux démontages et remontages de la partition.

Attention, l'utilitaire utilisateur de formatage de partition doit aussi être modifié. Si côté noyau, vous disposez de toutes les fonctions nécessaires à la manipulation des UUID, celles-ci ne sont pas disponibles côté utilisateur. Nous vous conseillons d'utiliser la bibliothèque `libuuid`, qui fait partie de la collection d'utilitaires standard `util-linux`. Il existe de multiples manières de se la procurer, elle est en particulier disponible sous `debian` dans le paquet `uuid-dev` et sous `archlinux` dans le paquet `util-linux`.

Étape 5 – Enrichissement de vos utilitaires de débogage

Vous enrichirez votre `sysfs`, en ajoutant au fichier l'UUID de la partition considérée. Puis, vous ferez de même avec l'`ioctl`, de façon à avoir l'UUID et le numéro d'inode de chaque lien distant.

Partie 3 : Liens physiques inter-partitions (dits liens distants)

Cette partie est un peu plus avancée et elle nécessitera de modifier un peu plus en profondeur le code de `ouichefs`. Elle suppose, naturellement, d'avoir correctement terminé les deux parties précédentes.

Pour le moment, les liens physiques font implicitement référence à des inodes positionnées sur la même partition (dépendantes du même superbloc). On veut maintenant implémenter un système qui permette à un lien physique de faire référence à une inode placée sur une autre partition `ouichefs`. Dans toute la suite, par abus de langage, on appellera un tel lien, un lien "distant" (même si les deux partitions sont bien évidemment sur la même machine).

Pour implémenter cette nouvelle fonctionnalité, vous allez tout d'abord ajouter un bit dans la structure inode qui permettra de différentier les liens distants. Attention, vous ne devez pas agrandir la taille des inodes de `ouichefs`.

Puis vous utiliserez le bloc d'index pour enregistrer la référence de l'inode originale sur la partition distante. Ainsi, dans le cas d'un lien distant, le bloc d'index contiendra l'UUID de la partition visée, suivi du numéro de l'inode visée.

Étape 6 – Création de liens distants vers des fichiers d'autre partition `ouichefs`

Dans un premier temps, modifiez toutes les structures internes pour permettre l'implémentation des liens distants. Puis, modifiez `ouichefs` pour permettre la création et la destruction de lien distant.

Vous pourrez aussi compléter votre `sysfs` pour afficher pour chaque partition des statistiques sur ce nouveau type de liens et ainsi vous aider à déboguer la suite de ce projet.

Étape 7 – Permettre l'utilisation de liens distants

Pour permettre des entrées/sorties sur ces liens distants, il suffit alors de modifier l'ouverture du fichier pour que la `struct file` et la `struct dentry` pointe sur l'inode original et non sur l'inode distante. Ainsi, si un lien L est créé vers un fichier F existant sur une autre partition `ouichefs`, l'inode pointée par ces structures sera celle du fichier F et non celle du fichier L. Il est facile d'obtenir des entrées/sorties sur ce nouveau type de lien.

Vous allez donc devoir trouver comment modifier le comportement du `open` et l'adapter lorsqu'il s'agira d'un lien distant, en modifiant la bonne fonction dans `ouichefs`.

Pour simplifier, vous pouvez considérer que toutes les partitions `ouichefs` seront toujours montées, ou mieux encore retourner une erreur si ce n'est pas le cas.

Partie 4 : Gestion de l'espace libre et migration d'inodes

Nous allons donc maintenant nous intéresser à la suppression des fichiers. Normalement à cette étape du projet, vous devriez pouvoir supprimer de manière transparente des liens physiques, qu'ils soient locaux ou vers une autre partition. Mais puisque le mécanisme de suppression est agnostique à la localité, on peut n'avoir plus que des liens distants vers un fichier (tous les liens physiques sur la partition originelle ont été supprimés). En pratique, cela peut poser des problèmes de gestion de l'espace disque : une partition pourrait être complète alors qu'elle n'a plus de fichier.

Étape 8 – Migration à la suppression lorsqu'il n'y a qu'un lien distant

Modifier le mécanisme de suppression des fichiers de façon à faire migrer tout le contenu du fichier, s'il ne reste plus de liens locaux, vers la partition qui contient un lien physique vers ce fichier. Ce dernier devient alors un lien physique standard. Pour simplifier, on supposera dans un premier temps qu'il n'y ait qu'un seul lien physique distant.

Étape 9 – Migration à la suppression

Vous pouvez maintenant lever la restriction sur le nombre de liens distants, en mettant à jour ceux qui n'auraient pas été choisis pour la migration. Pour guider le choix de la migration, on préférera la partition qui a le plus de blocs libres disponibles.

Étape 10 – Migration pour optimiser l'espace de stockage (Bonus pour l'été)

Puisque l'on a maintenant un mécanisme complet de migration, on peut chercher à résoudre le cas où l'on voudrait étendre un fichier à partir d'un lien distant alors que la partition qui l'héberge n'a plus d'espace disponible. L'idée est alors de faire migrer (si l'espace libre le permet) le fichier vers la partition qui contient le lien distant et permettre ainsi l'écriture demandée. Attention, cela n'est pas si simple, car au moment du `write` il n'est pas facile de savoir qu'il s'agit d'un lien distant.

