ENSEA
Ecole Nationale
Supérieure
de l'Electronique
et de ses Applications

3e année – IS – Systèmes et Réseaux

TP 3 – Réalisation d'un Système de Fichiers simple

D'après une idée de D. PICARD
C. BARÈS

Objectifs

Le but de ce TP est de développer un module du noyau capable de réaliser les opération de base d'un système de fichier simpliste (monter, lire un dossier, créer lire et écrire un fichier).

Principe

Notre système de fichier très simple va s'appeler **futosfs** (pour *«FUse Tiny OS FileSystem»*). Il contient au maximum 32 blocs de 4ko. La répartition des blocs est la suivante :

- 1 superblock
- 1 bloc d'inodes
- 1 bloc root
- les 29 blocs restant en données

Pour chaque inode, on associe un seul bloc. La taille maximale d'un fichier est donc d'un bloc. Il en découle que le nombre maximal de fichiers sur ce système de fichiers est de 29 (32 – le superblock, le bloc d'inodes et le bloc pour root).

Le superblock contient les information suivante :

- Un nombre magique (pour identifier le système de fichiers)
- Une bitmap des blocs
- Une bitmap des inodes
- La taille des blocs
- Le nombre de bloc (doit être 32)
- Le nombre d'inodes (doit être compris entre 1 et 29)
- Le numéro de l'inode root (doit être 1)

Notre structure d'inode est très simplifiée et contient les champs suivants :

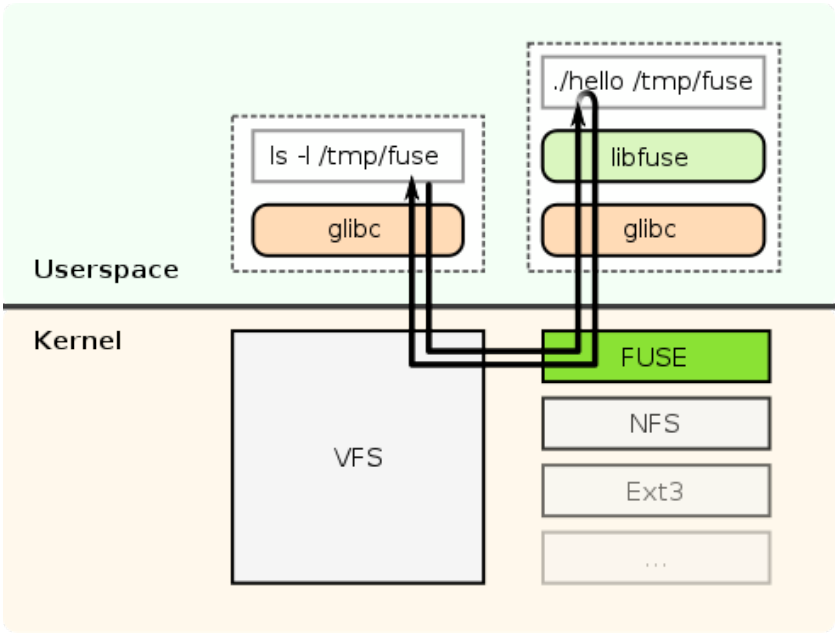
- Le numéro d'inode
- Le numéro du bloc où sont stockées les données
- diverses informations (les droits, *etc.*)

Une inode correspondant à un répertoire contient dans son bloc une suite de structure dentry collées les unes derrière les autres.

Notre structure dentry est aussi très simple et ne contient que deux champs : le numéro de l'inode associée au fichier et le nom du fichier.

Filesystem in UserSpacE (FUSE)

Afin de faciliter le développement de ce système de fichier, nous allons utiliser FUSE. Ce module noyau fonctionne de la manière suivante :



FUSE se comporte comme un pont entre l'espace utilisateur et le système de fichier : il permet de développer les fonctionnalités du système de fichiers directement depuis l'espace utilisateur, sans droit particulier.

Exemple

En guise d'échauffement, effectuez les tâches suivantes :

1. Récupperez les codes exemples disponibles sur moodle ;
2. Installez les sur Kerosen ou sur Tesla (`scp fichier login@kerosen:`)
3. Compilez le fichier `hello.c` à l'aide de :

```
$ gcc -Wall hello.c `pkg-config fuse --cflags --libs` -o hello
```

4. Si tout c'est bien passé, créez un répertoire dans tmp : `mkdir /tmp/futosfs`[1]
5. Montez votre système de ficher `hello` avec la commande :

```
$ ./hello /tmp/futosfs -d
```

   Le `-d` n'est pas obligatoire mais va vous faciliter le débogage.
6. Dans un nouveau terminal, explorer votre nouveau système de fichier avec `stat`, `ls`, `cat`...
7. Une fois terminé, démontez le système de fichier avec la commande :

```
$ fusermount -u /tmp/futosfs
```

   ou `<ctrl+c>` si vous êtes en mode débogage.

[1] Ne montez pas le système de fichier dans votre répertoire courant, NFS ne cohabite pas très bien avec FUSE (vu qu'ils utilisent tout les 2 le VFS).

Implémentation

Une archive contenant un squelette de code est fournie sur moodle. Cette archive contient un fichier binaire contenant un système de fichiers de test, ainsi qu'un fichier header contenant les structures du système de fichier et des fonctions utilitaires.

Il vous est fortement conseiller de lire les fichers d'exemple fournis (issu du code source de libfuse), ainsi que d'explorer le site doxygen : http ://libfuse.github.io/doxygen/, notamment les struture de données `fuse_operations` et `fuse_lowlevel_ops`.

Remarque : L'objectif de ce TP étant de coder un filesystem au niveau de ses inodes, il est donc plus simple d'utiliser l'API "low_level" de fuse. L'API "high_level" est plutôt réservée pour des applications où il n'y a pas besoin d'inodes (mais seulement des noms de fichiers), comme par exemple pour un systeme de fichier en réseau.

L'ordre conseillé de développement des fonctionnalités est le suivant :

1. en utilisant mmap pour charger le filesystem en mémoire, mapper les structures issues du fichier `tosfs.h` et afficher les informations sur le filesystem.
2. ensuite, reprendre le main de l'exemple "hello_ll.c", et fournir votre propre structure fuse_lowlevel_ops.
3. `getattr` afin de lire le contenu d'un répertoire (vérifier avec `stat`)
4. `readdir` afin de lire le contenu d'un répertoire (vérifier avec `ls -la`)
5. `lookup` afin de rechercher une entrée dans un répertoire (vérifier en créant un fichier avec `touch`)
6. `read` pour lire vos fichiers (`cat`)
7. `create` et `dir_add` afin de pouvoir créer un fichier (avec `touch`)
8. `write` écrire dans vos fichiers (utilisez `echo`)

Pour le débogage, en plus de l'option `-d` précédemment citée, vous pouvez utiliser `strace`, qui retrace l'ensemble des appels systèmes d'un processus. Par exemple :

```
$ LC_ALL=C strace cat /tmp/test_fuse/hello
```

L'utilisation du `LC_ALL=C` avant `strace` déactive la traduction des messages d'erreur en français, ce qui provoque encore plus d'appels système.