

Nom : AIDER

Prénom : Smail

N°Etudiant : 3603379

Parcours : M1-SAR - 2018/2019

Responsables : M Julien Sopena, Redha Gouicem

Compte-Rendu Projet - PNL
ouichefs – a simple educational filesystem for Linux

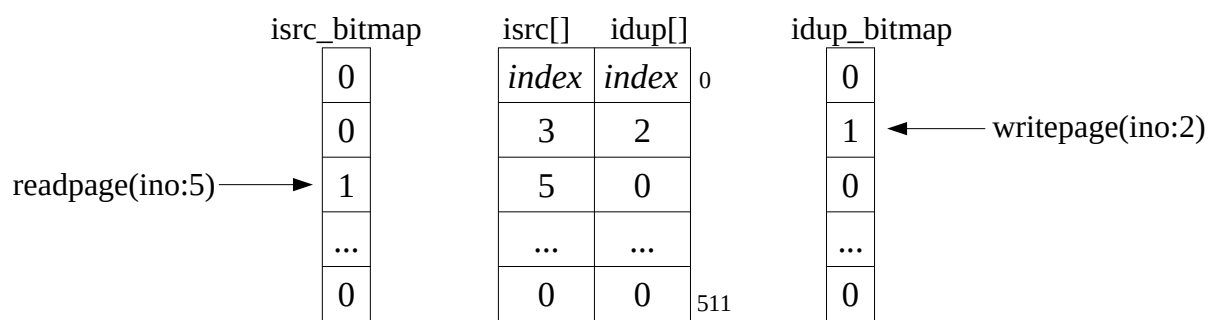
1) Dé-duplication de blocs

La dé-duplication de blocs consiste à détecter les blocs identiques sur la partition pour n'en conserver qu'un seul exemplaire. J'ai choisi l'approche **hors ligne** qui consiste à parcourir toute la partition pour trouver des doublons et cela au moment du **montage**.

- Implémentation :

Pour éviter de parcourir toute la partition à la recherche des doublons, j'ai fait en sorte de tracer toutes les lectures et écritures de pages en veillant à faire moins de traitements possibles entre les deux opérations(r/w).

Cette approche consiste à enregistrer, lors d'une lecture/écriture d'une page, le numéro de l'inode concerné. Le schéma ci-dessous explique le mécanisme implémenté :



Les deux tableaux **isrc[]** et **bitmap[]** sont stockés dans un bloc dont le numéro est enregistré dans la structure **struct ouichefs_sb_info.index_dupblock**. Ils sont indexés par **index** et ont une taille de 512.

Les deux bitmaps **isrc_bitmap** et **idup_bitmap**, allouées en mémoire, servent à éviter les doublons au niveau de chacun des tableaux **isrc[]** et **idup[]** et prennent en compte **tout** les inodes qui sont sur disque.

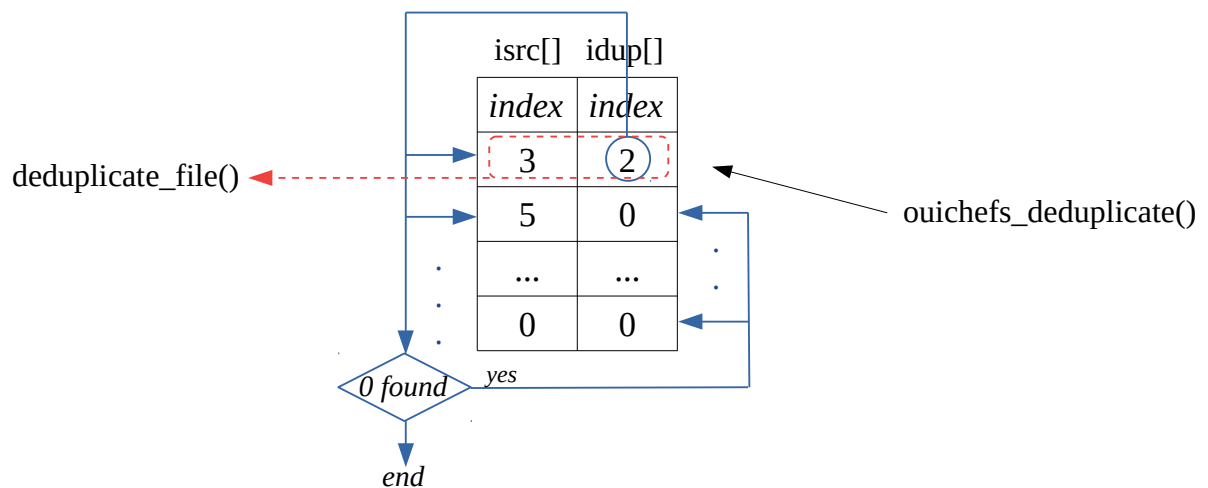
Ces éléments sont alloués dans la fonction **ouichefs_fill_super()**. On peut penser à chaîner d'autres blocs en cas de besoin, ce qui n'est pas implémenté dans la version actuelle.

Les fonctions utilisées sont :

- **dupblock_add_isrc()** : Ajoute l'inode 5 dans la liste **isrc[]**
- **dupblock_add_idup()** : Ajoute l'inode 2 dans la liste **idup[]**

- Utilisation :

Pour chaque inode de la liste **idup[]** dont les blocs de données sont susceptibles d'être dupliqués, on essaye de trouver leurs correspondants(même contenu) dans la liste des inodes **isrc[]** accédés en lecture. Si aucun résultat n'est trouvé, on refait une recherche dans la liste **idup[]**. Voir le schéma juste après.



Les fonctions utilisées sont :

- ***ouichefs_deduplicate()*** : Itère sur les deux liste isrc[] et idup[] et invoque *deduplicate_file()*
- ***deduplicate_file()*** : Itère sur l'*index_block* des deux inodes à la recherche de blocs similaires

Avant de démonter le système de fichiers, il faut sauvegarder le bloc qui stocke les deux tableaux sur disque et cela est fait dans la fonction *ouichefs_sync_fs()*. Concernant les bitmaps, elles sont libérées au moment du démontage dans la fonction *ouichefs_put_super()*.

2) Copie sur écriture

La copie sur écritures(copy-on-write ou cow) consiste à dupliquer les blocs partagés par plusieurs fichiers lors d'une modification(écriture). Pour réaliser cette fonctionnalité, il est nécessaire de savoir combien de fichiers partagent un bloc.

Pour cela, j'ai ajouté un emplacement dans le système de fichiers(**block store**) qui sert à stoker, pour chaque bloc sur disque, un compteur de référence(**uint32_t b_nlink**) stocké dans la structure **struct ouichefs_block_info**.

Les modifications apportées pour implémenter ce mécanisme sont indiquées dans l'annexe dans la partie **6. mkfs-ouichefs.c**.

```
+-----+-----+-----+-----+-----+-----+
| superblock | inode store | inode free bitmap | block free bitmap | block store | data blocks |
+-----+-----+-----+-----+-----+-----+
```

Les fonctions qui manipulent ce champ sont :

- **ouichefs_link_block()**
- **ouichefs_unlink_block()**

Ces fonctions prennent en paramètres le *super_block* et le *numéro de bloc* concerné. Elle récupèrent l'entrée correspondante dans le block store et incrémente/décrémente le compteur de référence. Si ce dernier atteint zéro, on libère le bloc(*put_block*).

Lors d'un appel système *write()*, la fonction **ouichefs_write_begin()** est appelée avant que les données soient écrites dans le page cache. Cette fonction vérifie si l'écriture demandée peut être satisfaite et alloue les blocs nécessaires. Elle reçoit comme arguments la position dans le fichier(*pos*) et la taille des données à écrire(*len*).

Avec ses informations(*pos* et *len*), on peut déterminer les blocs concernés par l'écriture. C'est dans la fonction *ouichefs_write_begin()* qu'il faudrait donc vérifier le compteur de référence de ses blocs et les dupliquer si sa valeur est supérieur strictement à 1. Cette vérification est faite par la fonction **ouichefs_cow()**.

Le champ *b_nlink* est manipulé au moment de :

- L'allocation d'un bloc *ouichefs_file_get_block()*
- La suppression d'un fichier *ouichefs_unlink()*
- La dé-duplication d'un fichier *deduplicate_file()*
- La duplication de blocs *ouichefs_cow()*

Remarque – dé-duplication au démontage de la partition :

Lors de la déduplication, il faut s'assurer que les pages dirty concernées par le COW soient bien écrites sur disque car la fonction de dé-duplication lit les blocs de données depuis le disque.

On aimerait bien qu'une page modifiée dans le page cache et dont l'écriture n'a pas été reportée sur disque soit quand même prise en compte et ne pas être dé-dupliquée au démontage.

Ce problème n'est pas présent si la déduplication est faite au montage de la partition car le page cache est désormais vide.

3) Annexe

La liste des fonctions/structures modifiées[*] / ajoutées[+] :

	line number :
1. util.c, util.h	
1.1. + ouichefs_cow()	[20]
1.2. + ouichefs_link_block()	[114]
1.3. + ouichefs_unlink_block()	[149]
1.4. + deduplicate_file()	[190]
1.5. + ouichefs_deduplicate()	[273]
1.6. + dupblock_add_isrc()	[356]
1.7. + dupblock_add_idup()	[389]
2. ouichefs.h	
2.1. + struct ouichefs_block_info	[59]
2.2. + #define OUCHEFS_BINFO_PER_BLOCK	[66]
2.3. + #define OUCHEFS_FIRST_DT_BLOCK	[69]
2.4. + #define OUCHEFS_FIRST_BINFO_STORE	[73]
2.5. + #define OUCHEFS_BINFO_INDEX	[77]
2.6. + struct ouichefs_dup_block	[113]
2.7. * struct ouichefs_sb_info	[93:96]
3. file.c	
3.1. * ouichefs_file_get_block()	[62]
3.2. * ouichefs_readpage()	[85]
3.3. * ouichefs_writepage()	[100]
3.4. * ouichefs_write_begin()	[136]
4. super.c	
4.1. * ouichefs_write_inode()	[89]
4.2. * ouichefs_put_super()	[104, 105]
4.3. * ouichefs_sync_fs()	[130, 131, 175:193]
4.4. * ouichefs_fill_super()	[265, 266, 319:368]
5. inode.c	
5.1. * ouichefs_unlink()	[370]
6. mkfs_ouichefs.c	
6.1. * struct ouichefs_super_block	[50, 51, 53]
6.2. * write_super_block()	[110, 111, 122]
6.3. * write_inode_store()	[167]
6.4. * write_bfree_block()	[257]
6.5. + write_bstore_block()	[305]
6.6. * main()	[432]

4) Résultats

1. Déduplication

- 1.1. Test_01 : Créer un fichier **file** et lui faire une copie **copy** avec \$ cp :
Même résultat avec la création de deux fichiers identiques.

The terminal window shows the following commands and output:

```
[root@pnl-tp sysfs]# df -h -k .
Filesystem      1K-blocks  Used Available Use% Mounted on
/dev/loop0      102400    1128   101272   2% /root/share/pnl/mkfs/sysfs

[root@pnl-tp sysfs]# echo hello > file
[ 2332.463356] -> ouichefs_cow
[ 2332.465546] -> ouichefs_link_block
[ 2332.468684] -- block_count[283 -> 1] (binfo: 255, shift: 2)

[root@pnl-tp sysfs]# cp file copy
[ 2336.198024] -> ouichefs_cow
[ 2336.199579] -> ouichefs_link_block
[ 2336.200481] -- block_count[285 -> 1] (binfo: 255, shift: 4)

[root@pnl-tp sysfs]# df -h -k .
Filesystem      1K-blocks  Used Available Use% Mounted on
/dev/loop0      102400    1144   101256   2% /root/share/pnl/mkfs/sysfs

[root@pnl-tp sysfs]# cd ../
[root@pnl-tp mkfs]# umount sysfs/
[ 2347.302842] ouichefs: unmounted disk

[root@pnl-tp mkfs]# mount test.img sysfs/
[ 2350.128204] -> deduplicate_file
[ 2350.128995] -> ouichefs_unlink_block
[ 2350.129594] -- block_count[283 -> 0]
[ 2350.130124] -> ouichefs_link_block
[ 2350.130712] -- block_count[285 -> 2] (binfo: 255, shift: 4)
[ 2350.131449] -- src_block[285] <-- dup_block[283]
[ 2350.132136] -> deduplicate_file
[ 2350.132619] --> 1 blocks deduplicated <--
[ 2350.133149] ouichefs: '/dev/loop0' mount success

[root@pnl-tp mkfs]# cd sysfs/
[root@pnl-tp sysfs]# df -h -k .
Filesystem      1K-blocks  Used Available Use% Mounted on
/dev/loop0      102400    1144   101256   2% /root/share/pnl/mkfs/sysfs
```

Annotations:

- Deux fichiers identiques dont les blocs sont différents :**
 - file1 → 283
 - copy → 285
- $1128 + 4 + 4 = 1144$
data block : 4KiB
index_block : 4KiB
- Bloc 283 libéré.
-Incrément de nlink du bloc 285.
→ Le fichier dont le bloc = 283 référence le bloc 285.
→ 1 bloc libéré
- L'espace occupée reste inchangée.

1.2. Test_02 : Copie d'un fichier plus grand avec 3 blocs de données :

```

Terminal
File Edit View Search Terminal Tabs Help

Terminal x Terminal x

Filesystem      1K-blocks  Used Available Use% Mounted on
/dev/loop0      102400    1144    101256    2% /root/share/pnl/mkfs/sysfs
[root@pnl-tp sysfs]# cp ../mkfs-ouichefs.c .
[ 6273.077955] -> ouichefs_cow
[ 6273.079441] -> ouichefs_link_block
[ 6273.082023] -- block_count[286 -> 1] (binfo: 255, shift: 5)
[ 6273.085405] -> ouichefs_cow
[ 6273.086996] -> ouichefs_link_block
[ 6273.088846] -- block_count[287 -> 1] (binfo: 255, shift: 6)
[ 6273.091671] -> ouichefs_cow
[ 6273.093271] -> ouichefs_link_block
[ 6273.095188] -- block_count[288 -> 1] (binfo: 255, shift: 7)
[root@pnl-tp sysfs]# cp mkfs-ouichefs.c copy.c
[ 6280.978982] -> ouichefs_cow
[ 6280.980589] -> ouichefs_link_block
[ 6280.983249] -- block_count[290 -> 1] (binfo: 255, shift: 9)
[ 6280.985951] -> ouichefs_cow
[ 6280.987831] -> ouichefs_link_block
[ 6280.988536] -- block_count[291 -> 1] (binfo: 255, shift: 10)
[ 6280.989618] -> ouichefs_cow
[ 6280.990206] -> ouichefs_link_block
[ 6280.990824] -- block_count[292 -> 1] (binfo: 255, shift: 11)
[root@pnl-tp sysfs]# df -h -k .
Filesystem      1K-blocks  Used Available Use% Mounted on
/dev/loop0      102400    1176    101224    2% /root/share/pnl/mkfs/sysfs
[root@pnl-tp sysfs]# cd ../
[root@pnl-tp mkfs]# umount sysfs/
[ 6293.504107] ouichefs: unmounted disk
[root@pnl-tp mkfs]# mount test.img sysfs/
[ 6297.092566] -> deduplicate_file
[ 6297.093270] -> ouichefs_unlink_block
[ 6297.093859] -- block_count[286 -> 0]
[ 6297.094417] -> ouichefs_link_block
[ 6297.095195] -- block_count[290 -> 2] (binfo: 255, shift: 9)
[ 6297.096059] -- src_block[290] <-- dup_block[286]
[ 6297.096989] -> ouichefs_unlink_block
[ 6297.097585] -- block_count[287 -> 0]
[ 6297.098063] -> ouichefs_link_block
[ 6297.098586] -- block_count[291 -> 2] (binfo: 255, shift: 10)
[ 6297.099323] -- src_block[291] <-- dup_block[287]
[ 6297.100136] -> ouichefs_unlink_block
[ 6297.100594] -- block_count[288 -> 0]
[ 6297.101065] -> ouichefs_link_block
[ 6297.101621] -- block_count[292 -> 2] (binfo: 255, shift: 11)
[ 6297.102360] -- src_block[292] <-- dup_block[288]
[ 6297.103054] -> deduplicate_file
[ 6297.103544] --> 3 blocks deduplicated <--
[ 6297.104081] ouichefs: '/dev/loop0' mount success
[root@pnl-tp mkfs]# cd sysfs/
[root@pnl-tp sysfs]# df -h -k .
Filesystem      1K-blocks  Used Available Use% Mounted on
/dev/loop0      102400    1176    101224    2% /root/share/pnl/mk
[root@pnl-tp sysfs]#

```

-Le fichier mkfs-ouichefs.c occupe 3 blocs de données (286, 287, 288).
-La copie de ce fichier copy.c occupe aussi 3 blocs de données (290, 291, 292).

- Le nlink de ses blocs vaut 1 et il est stocké dans le bloc 255 dans l'entrée 5, 6, ...

Les blocs 286, 287 et 288 sont libérés et le fichier référence les blocs 290, 291 et 292 avec un nlink qui vaut 2.

→ Au total : 3 blocs libérés.

L'espace occupée reste inchangé.

2. Copy-On-Write

2.1. Modification du fichier **copy** de l'exemple 1.1.Test_01 avec \$ echo

```
[root@pnl-tp sysfs]# ls
copy copy.c file mkfs-ouichefs.c
[root@pnl-tp sysfs]# df -h -k .
Filesystem      1K-blocks  Used Available Use% Mounted on
/dev/loop1      102400    1176   101224    2% /root/share/pnl/mkfs/sysfs
[root@pnl-tp sysfs]# cat file
hello
[root@pnl-tp sysfs]# cat copy
hello
[root@pnl-tp sysfs]# echo bonjour > copy
[10563.275922] -> ouichefs_cow
[10563.277835] -- old_block[285] -> new_block[283]
[10563.280020] -> ouichefs_unlink_block
[10563.281059] -- block_count[285 -> 1]
[root@pnl-tp sysfs]# cat file
hello
[root@pnl-tp sysfs]# cat copy
bonjour
[root@pnl-tp sysfs]# sync
[root@pnl-tp sysfs]# cd ../
[root@pnl-tp mkfs]# umount sysfs/
[10579.343456] ouichefs: unmounted disk
[root@pnl-tp mkfs]# mount test.img sysfs/
[10581.803062] -> deduplicate_file
[10581.803781] --> 0 blocks deduplicated <--
[10581.804295] ouichefs: '/dev/loop1' mount success
[root@pnl-tp mkfs]# cd sysfs/
[root@pnl-tp sysfs]# cat file
hello
[root@pnl-tp sysfs]# cat copy
bonjour
[root@pnl-tp sysfs]# df -h -k .
Filesystem      1K-blocks  Used Available Use% Mounted on
/dev/loop1      102400    1176   101224    2% /root/share/pnl/mkfs/sysfs
[root@pnl-tp sysfs]#
```

Les deux fichiers ont le meme contenu

-En modifiant le contenu du fichier copy, il y a une allocation d'un nouveau bloc 283 et une décrémentation du compteur de référence du bloc 285.

-Après démontage et remontage de la partition, pas de déduplication faite.

-Les deux fichiers ont toujours un contenu différent.

L'espace occupée reste inchangé.

2.2. Modification du fichier **copy** de l'exemple 1.2.Test_02 avec **concaténation**

```
[root@pnl-tp sysfs]# ls
copy copy.c file mkfs-ouichefs.c
[root@pnl-tp sysfs]# df -h -k .
Filesystem      1K-blocks  Used Available Use% Mounted on
/dev/loop0      102400    1176    101224   2% /root/share/pnl/mkfs/sysfs
[root@pnl-tp sysfs]# cat file
hello
[root@pnl-tp sysfs]# cat copy
hello
[root@pnl-tp sysfs]# echo world >> copy
[ 1542.138916] -> ouichefs_cow
[ 1542.140294] -> ouichefs_link_block
[ 1542.142868] -- block_count[283 -> 1] (binfo: 255, shift: 2)
[ 1542.143760] -- old_block[285] -> new_block[283]
[ 1542.144485] -> ouichefs_unlink_block
[ 1542.145192] -- block count[285 -> 1]
[root@pnl-tp sysfs]# cat file
hello
[root@pnl-tp sysfs]# cat copy
hello
world
[root@pnl-tp sysfs]# sync
[root@pnl-tp sysfs]# cd ../
[root@pnl-tp mkfs]# umount sysfs/
[ 1573.397486] ouichefs: unmounted disk
[root@pnl-tp mkfs]# mount test.img sysfs/
[ 1576.681095] -> deduplicate_file
[ 1576.681750] -> ouichefs_unlink_block
[ 1576.682288] -- block_count[283 -> 0]
[ 1576.682774] -> ouichefs_link_block
[ 1576.683449] -- block_count[285 -> 2] (binfo: 255, shift: 4)
[ 1576.684170] -- src_block[285] <-- dup_block[283]
[ 1576.684946] --> 1 blocks deduplicated <--
[ 1576.685489] ouichefs: '/dev/loop0' mount success
[root@pnl-tp mkfs]# cd sysfs/
[root@pnl-tp sysfs]# cat file
hello
[root@pnl-tp sysfs]# cat copy
hello
[root@pnl-tp sysfs]#
```

Les deux fichiers partagent le **même** contenu

Allocation d'un nouveau bloc 283 pour le fichier modifié

Les deux fichiers ont un contenu **différent**

Après démontage et remontage, on remarque une **déduplication** du bloc 283 qui survient

Le contenu des deux fichiers est à **nouveau identique !**

→ Il y a un problème lors d'une modification **partielle** d'un fichier. En effet, la fonction `ouichefs_cow()` duplique le contenu du bloc 285 dans le bloc 283 avant que l'écriture soit faite. Juste après l'écriture et en lisant le contenu des deux fichiers, on remarque que la modification a bien été prise en compte dans le **page cache**. Normalement, après le démontage de la partition, le contenu du page cache devait être reporter sur disque. Sauf que, on montant la partition à nouveau, la fonction de déduplication indique que le contenu des blocs 283 et 285 est **identique !**