Nom : AIDER
Prénom : Smail
N°Etudiant : 3603379
Parcours : M1-SAR - 2018/2019
Responsables : M Julien Sopena, Redha Gouicem

SORBONNE UNIVERSITÉ

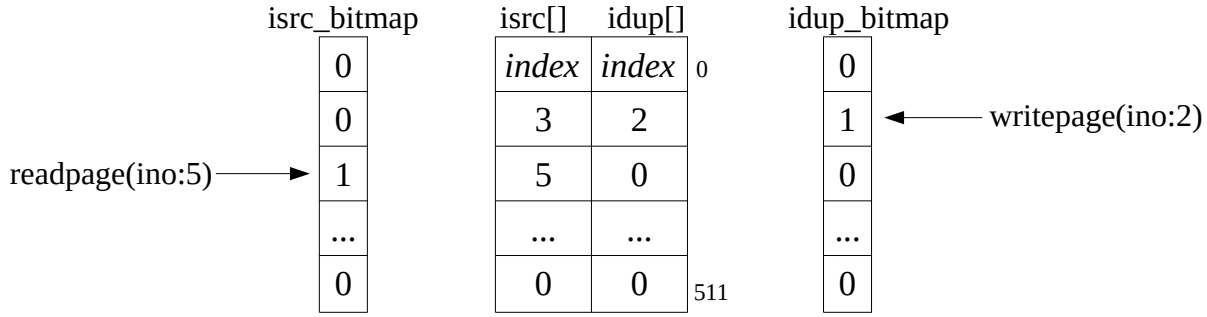
Compte-Rendu Projet - PNL
ouichefs – a simple educational filesystem for Linux

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 :



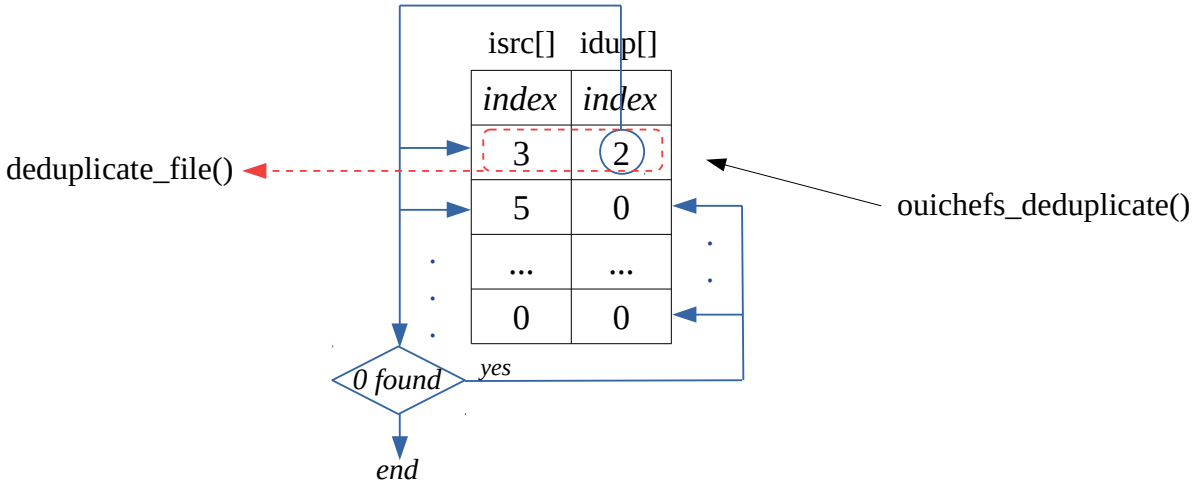
Les deux tableaux ***isrc[]*** et ***bitmap[]*** sont stockés dans un bloc dont le numéro est enregistré dans la structure ***struct ouichefs_sb_info.index_dupblock***. Ils sont indexés par *index* et ont une taille de 512.

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 élements sont alloués dans la fonction *ouichefs_fill_super()*. On peut penser à chainer 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.



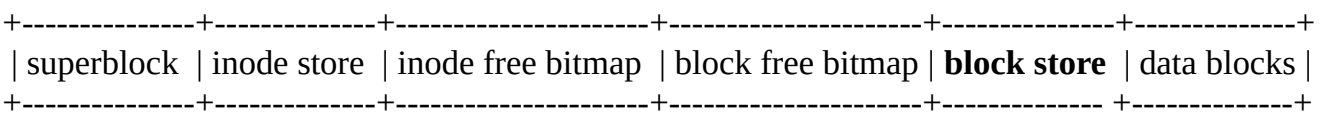
Les fonctions utilisées sont :
- ***ouichefs_deduplicate() :*** Itère sur les deux liste isrc[] et idup[] et invoque *deduplicate_file()*
- ***deduplicate_file() :*** Itère sur l'*index_block* des deux inodes à la recherche de blocs similaires

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 systeme 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***.



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 elle 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 reporter sur disque soit quand même prise en compte et ne pas être dé-dupliquer 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[+] :

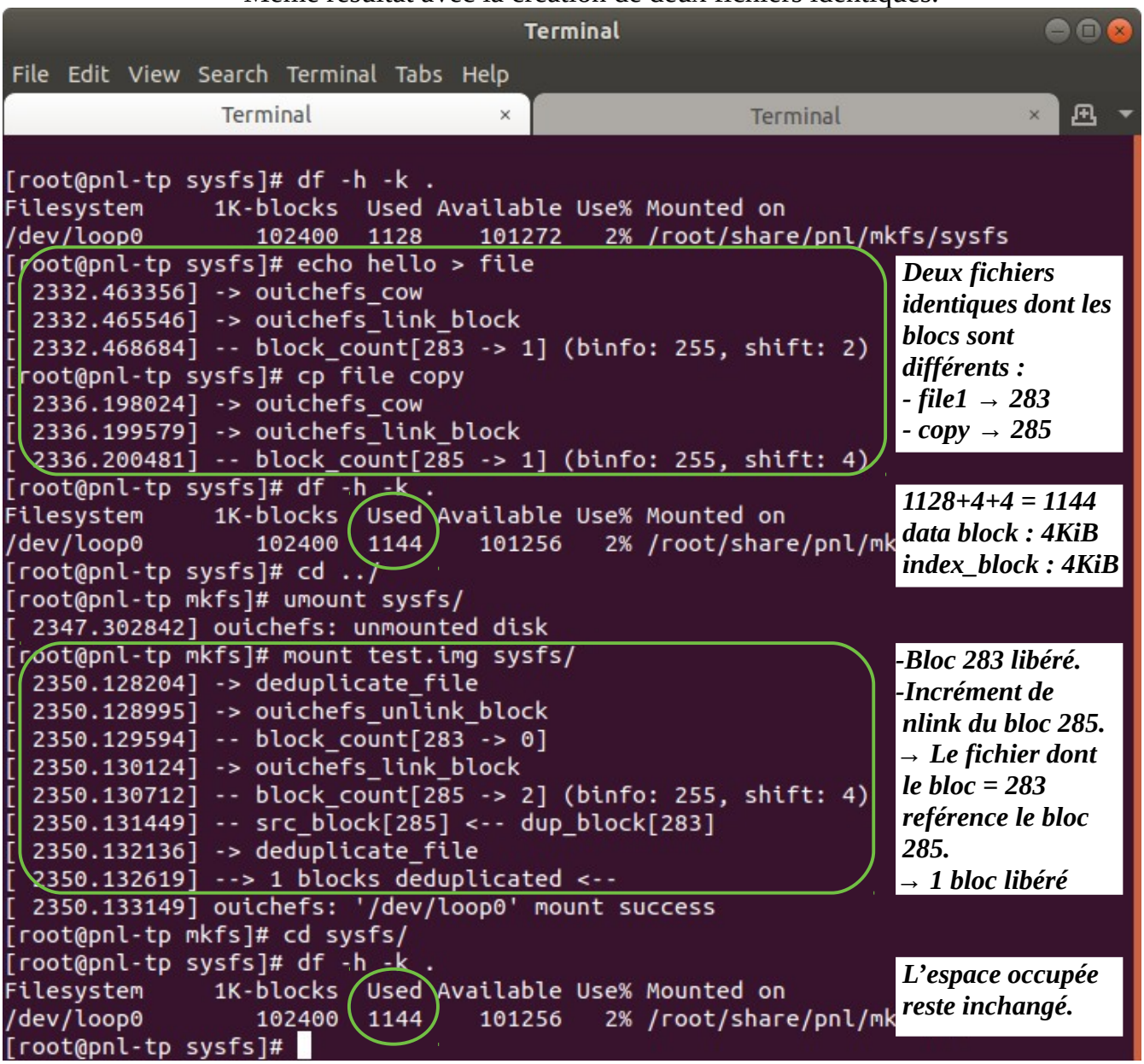
1.	**util.c, util.h**	**line number :**
1.1.	+ ouichefs_cow()	[20]
1.2.	+ ouichefs_link_block()	[114]
1.3.	+ ouichefs_unlink_block()	[149]
1.4.	+ deduplicate_file()	[190]
1.5.	+ ouichefs_deduplicate()	[273]
1.6.	+ dupblock_add_isrc()	[356]
1.7.	+ dupblock_add_idup()	[389]
2.	**ouichefs.h**	
2.1.	+ struct ouichefs_block_info	[59]
2.2.	+ #define OUICHEFS_BINFO_PER_BLOCK	[66]
2.3.	+ #define OUICHEFS_FIRST_DT_BLOCK	[69]
2.4.	+ #define OUICHEFS_FIRST_BINFO_STORE	[73]
2.5.	+ #define OUICHEFS_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 un copie **copy** avec $ cp :

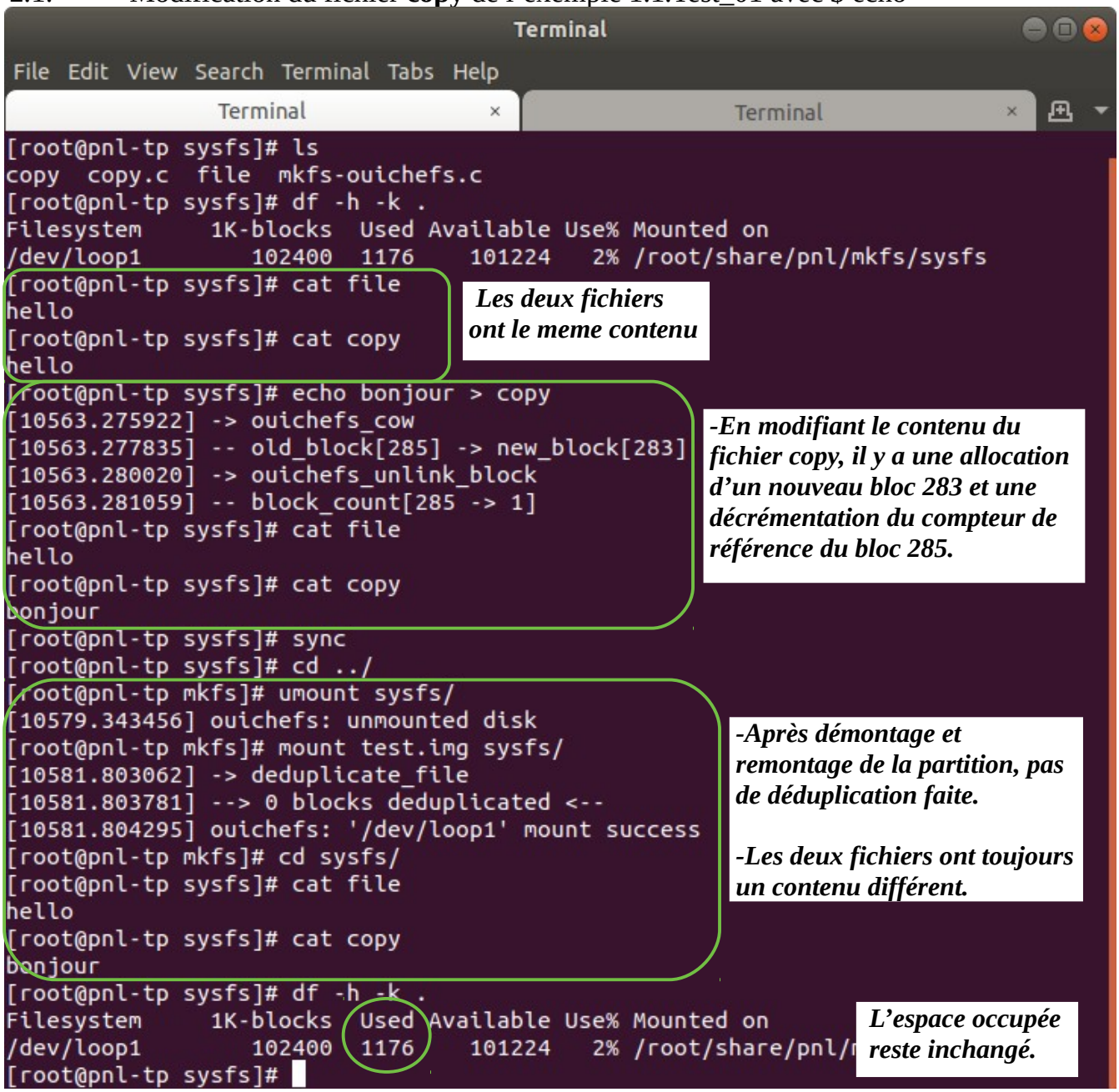
Même résultat avec la création de deux fichiers identiques.



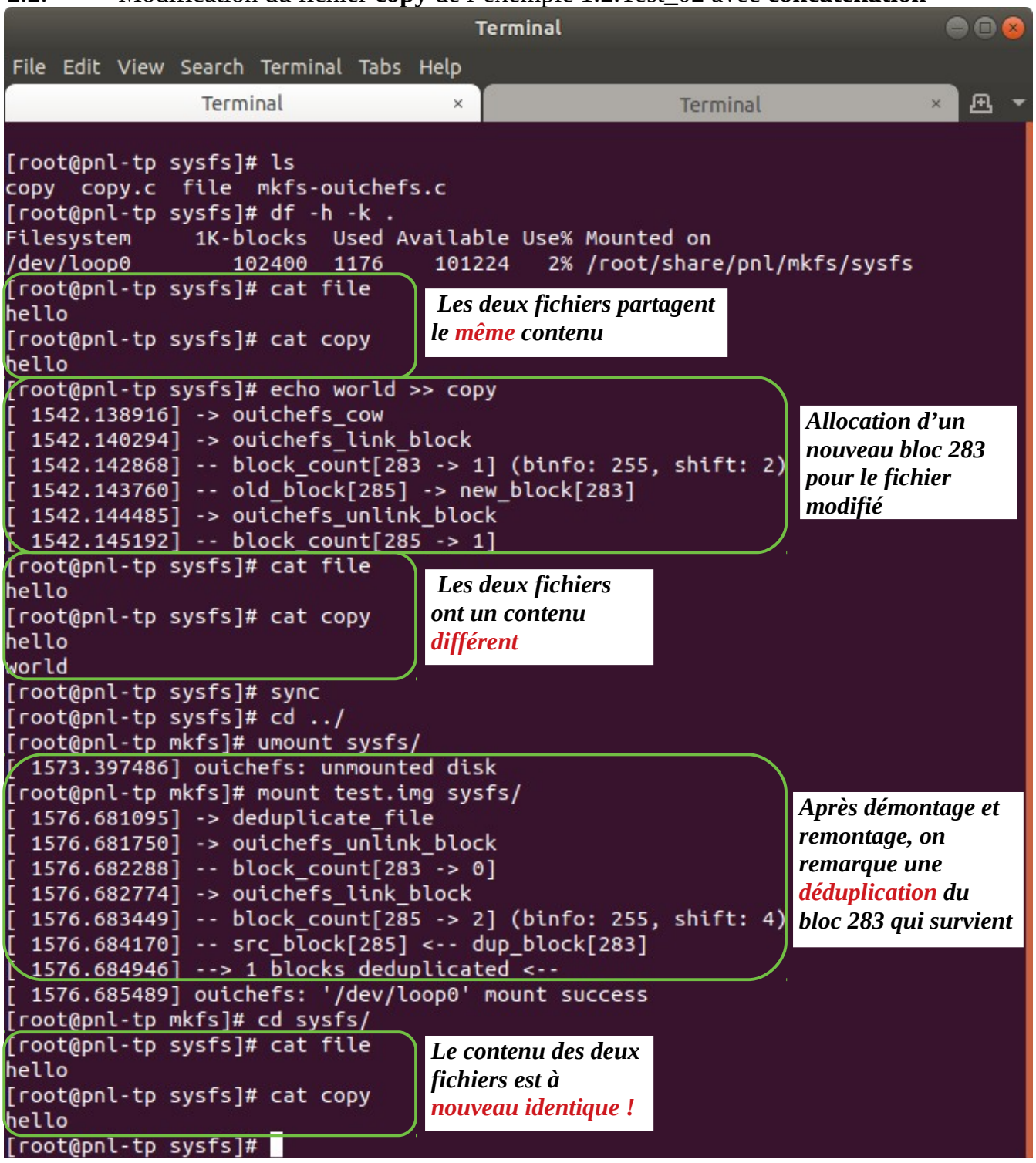
1.2. Test_02 : Copie d'un fichier plus grand avec 3 blocs de données :

2. Copy-On-Write

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



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



→ 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 !**