**Rapport de projet Pignoufs**

**Équipe**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nom | Prenom | N°Etudiant | Mail | Tag Moule |
| Xue | Guillaume | 22101031 | guillaume.xue@etu.u-paris.fr | @xue |
| Yu | David | 22110478 | david.yu@etu.u-paris.fr | @yu XiaoGunFr |

**Structure du système de fichiers**

Le système de fichiers “Pignoufs” repose sur une organisation stricte de blocs de 4096 octets, avec une vérification d’intégrité par SHA1. Chaque bloc a une structure dédiée selon son rôle (superbloc, bitmap, inode, données, etc.).

**Organisation globale**

Le fichier conteneur est découpé en 4 grandes zones contiguës :

1. Superbloc : un seul bloc décrivant les paramètres globaux.

2. Bitmaps : blocs décrivant l’état (libre ou alloué) de chaque bloc.

3. Inodes : un bloc par fichier (ou répertoire si extension).

4. Blocs allouables : utilisés pour stocker les données ou adresses.

Le nombre total de blocs est nb\_b = 1 + nb1 + nb\_i + nb\_a.

**Superbloc**

struct pignoufs {

  char magic[8];          // "pignoufs"

  int32\_t nb\_b;       // Total blocs

  int32\_t nb\_i;           // Inodes

  int32\_t nb\_a;           // Blocs allouables

  int32\_t nb\_l;           // Blocs libres

  int32\_t nb\_f;           // Inodes allouées (fichiers)

  char zero[3972];       // Remplissage à 4000 octets

  uint8\_t sha1[20];       // SHA1 du bloc

  uint32\_t type;          // Type = 1

  char padding[72];       // Zone disponible

};

**Bloc de bitmap**

struct bitmap\_block {

  uint8\_t bits[4000];     // 32000 bits (1 par bloc)

  uint8\_t sha1[20];       // SHA1 du contenu

  uint32\_t type;          // Type = 2

  char padding[72];       // Zone disponible

};

**Inode**

Chaque fichier possède une inode, occupant un bloc entier. Les blocs de données sont référencés directement ou via indirection.

struct inode {

  uint32\_t flags;                  // Bits : existence, droits, verrous, type

  uint32\_t file\_size;              // Taille du fichier

  uint32\_t creation\_time;

  uint32\_t access\_time;

  uint32\_t modification\_time;

  char filename[256];             // Nom du fichier

  int32\_t direct\_blocks[900];     // Blocs de données directs

  int32\_t double\_indirect\_block;  // Pointeur vers un bloc de type 7

  char extensions[120];           // Zone d’extension

  uint8\_t sha1[20];               // SHA1 du contenu

  uint32\_t type;                  // Type = 3

  int32\_t profondeur;            // Profondeur (si sous-répertoires)

  char padding[68];              // Remplissage

};

**Bloc de données**

Contient 4000 octets de données, éventuellement suivis de remplissage pour le SHA1.

struct data\_block {

  char data[4000];

  uint8\_t sha1[20];

  uint32\_t type;    // Type = 5

  char padding[72];

};

**Bloc d’adresses**

Utilisé pour l’indirection simple (type 6) ou double (type 7).

struct address\_block {

  int32\_t addresses[1000]; // Adresses de blocs de données ou d’adresses

  uint8\_t sha1[20];

  uint32\_t type;           // Type = 6 ou 7

  char padding[72];

};

**Fonctionnalités implémentées**

|  |  |  |
| --- | --- | --- |
| Nom de fonction | But | Description |
| cmd\_mkfs | Crée un nouveau système de fichiers Pignoufs dans un fichier conteneur. | Initialise le superbloc, les bitmaps, les inodes et les blocs de données. |
| cmd\_ls | Liste le contenu d’un répertoire ou affiche les informations d’un fichier. | Affiche les fichiers/répertoires, avec ou sans détails (-l). |
| cmd\_tree | Affiche l’arborescence complète du système de fichiers. | Affichage récursif des dossiers/fichiers sous forme d’arbre. |
| cmd\_mkdir | Crée un répertoire dans le système de fichiers. | Prend en argument le chemin du dossier à créer. |
| cmd\_cat | Affiche le contenu d’un fichier interne sur la sortie standard. | Lit et affiche les données d’un fichier du FS. |
| cmd\_input | Écrit le contenu de l’entrée standard dans un fichier interne (remplace le contenu). | Crée le fichier si besoin, écrase le contenu existant. |
| cmd\_addinput | Ajoute le contenu de l’entrée standard à la fin d’un fichier interne. | Crée le fichier si besoin, ajoute à la suite du contenu existant. |
| cmd\_add | Ajoute le contenu d’un fichier externe à la fin d’un fichier interne. | Similaire à cmd\_addinput mais lit depuis un fichier externe. |
| cmd\_cp | Copie fichiers ou dossiers entre l’interne et l’externe, ou à l’intérieur du FS. | Gère : interne→externe, externe→interne, interne→interne (fichiers et dossiers). |
| cmd\_mv | Déplace ou renomme un fichier ou dossier interne. | Peut changer le nom ou déplacer dans un autre dossier. |
| cmd\_rm | Supprime un fichier interne. | Gère le verrouillage et la libération des blocs. |
| cmd\_rmdir | Supprime un dossier interne (récursivement). | Supprime tous les fichiers/sous-dossiers contenus. |
| cmd\_chmod | Modifie les permissions (lecture/écriture) d’un fichier interne. | Ajoute ou enlève les droits de lecture/écriture. |
| cmd\_lock | Place un verrou de lecture ou d’écriture sur un fichier interne. | Garde le verrou jusqu’à réception d’un signal. |
| cmd\_find | Recherche des fichiers selon le type, le nom ou la date. | Filtre par type (fichier/dossier), nom, et date (création, modification, accès). |
| cmd\_grep | Recherche un motif texte dans tous les fichiers internes. | Affiche les noms des fichiers contenant le motif. |
| cmd\_df | Affiche l’espace libre (blocs et inodes) du système de fichiers. | Indique le nombre de blocs et d’inodes libres. |
| cmd\_fsck | Vérifie l’intégrité du système de fichiers (type, bitmap, SHA1, etc.). | Multithreadé, vérifie la cohérence des blocs et des inodes. |

**Structure des dossiers dans Pignoufs**

Dans Pignoufs, un dossier est représenté par une inode classique, avec un bit de type activé dans flags :

(type << 5)  // Bit 5 : 1 = dossier, 0 = fichier

**Contenu du dossier : pointeurs vers des inodes enfants**

Chaque dossier contient jusqu’à 900 pointeurs directs (direct\_blocks[]) vers :

• des inodes de fichiers,

• ou des inodes d’autres dossiers (sous-répertoires).

Cela permet de construire une structure arborescente récursive.

**Profondeur**

Chaque inode contient un champ profondeur :

int32\_t profondeur; // Profondeur dans l'arborescence

Ce champ est mis à jour lors de la création de sous-dossiers. Il est utile pour :

• des fonctions comme tree, ls, find.

• éviter des erreurs de structure circulaire.

**Exemple de structure**

[inode "cours"]           (dossier, profondeur 0)

   └── direct\_blocks[0] → [inode "PSA"]       (dossier, profondeur 1)

         └── direct\_blocks[0] → [inode "tp1"] (fichier, profondeur 2)

**Threads de vérification SHA1**

Afin d’améliorer la robustesse et la réactivité du système de fichiers Pignoufs, nous avons mis en place un thread dédié à la vérification d’intégrité des blocs via SHA1. Ce mécanisme permet de paralléliser les lectures de blocs et leur vérification, comme suggéré dans le sujet.

**Architecture**

Nous avons implémenté une file de tâches (sha1\_queue\_t) dans laquelle les blocs à vérifier sont ajoutés par les fonctions d’accès au système de fichiers. Un thread consommateur exécute la fonction sha1\_worker, qui :

• attend qu’une tâche soit disponible via une condition (pthread\_cond\_wait),

• vérifie le SHA1 du bloc via la fonction check\_sha1,

• signale une erreur si l’empreinte ne correspond pas.

**Fonctions principales**

* **void \*sha1\_worker(void \*arg)** : Thread infini (arrêt sur signal done) qui consomme des tâches SHA1 à vérifier.
* **void check\_sha1(const void \*data, size\_t len, uint8\_t \*ref)** : Vérifie que le SHA1 calculé à partir de data et len correspond au SHA1 attendu (ref). En cas de divergence, un message explicite est affiché.
* **void calcul\_sha1(const void \*data, size\_t len, uint8\_t \*out)** : Fonction qui utilise l’API OpenSSL (EVP\_\*) pour calculer l’empreinte SHA1 de manière sécurisée et portable.

**Gestion de la concurrence**

Nous avons choisi un verrouillage par bloc, plus fin que le verrou global, permettant à plusieurs processus d’interagir avec des blocs différents du système de fichiers de manière concurrente.

**Fonctions utilisées**

* **int lock\_block(int fd, int64\_t offset, int lock\_type)** : lock\_block utilise l’appel système fcntl avec F\_RDLCK ou F\_WRLCK sur une plage de 4096 octets (la taille d’un bloc).
* **int unlock\_block(int fd, int64\_t offset)** : unlock\_block libère le verrou (F\_UNLCK).

Si le verrou est bloquant (F\_SETLKW), ce qui signifie qu’un processus attend s’il n’a pas accès immédiatement.

**Tests et validation**

Afin de garantir la conformité fonctionnelle et la robustesse du système de fichiers Pignoufs, nous avons mis en place un ensemble complet de tests automatisés

Chaque test vérifie :

• que la commande fonctionne sans erreur ;

• que le comportement observé est conforme à l’attendu ;

• que les fichiers ou dossiers sont correctement manipulés ;

• que les opérations échouent en cas d’erreur d’accès ou de droit.