Rapport du projet de système

Mise en place d'un système de gestion de fichiers basé sur linux

Réalisé par :

Iheb BESBES : IATIC3

Iyed CHEBERLI : IATIC3

sami FAKHFAKH : IATIC3

Seifeddine BEN JEMAA :IATIC3

Med Ilyes EL AJROUD : IATIC3

Med firas ESSOURI : IATIC3

Encadrement :

Mme Dhekra ABOUDA

Mr Thierry GARCIA

Sommaire

1. Introduction

“UNIX is basically a simple operating system, but you have to be a genius to understand the simplicity.”

Dennis Ritchie

UNIX, un système d’exploitation décrit “simple” par ses concepteurs, mais qui a laissé son empreinte sur pratiquement tous les aspects de nos systèmes d’exploitations modernes, nous étudiant parmi ces aspects les systèmes de gestions de fichiers sous linux, dans ce cadre un projet d'implémentation sgf nous été proposé, l’objectif principal est de modéliser les notions d’inodes et de blocs

Nous allons décrire dans ce rapport notre approche envers de ce sujet on décrivant;

Les besoins de l’utilisateur.

Le cahier des charges

Notre stratégie d'implémentation et prévision de structures de données

Et enfin notre méthodologie de travail d'équipe envers ce projet

2- Conception

2.1 BESOIN UTILISATEUR:

Nous prévoyons que l’utilisateur sera capable de d’effectuer des opérations de créations, modifications sur des fichiers stocké dans un environnement de notre propre conception à travers notre système.

Un utilisateur de notre système doit être capable de gérer les fichiers présent dans cet environnement en modifiant leurs emplacements, noms, droits d'accès ainsi qu’un contrôle de cet environnement commençant par la gestion des espaces libres, la modification de l'arborescence de répertoires, jusqu'à la réalisations des opérations de recherches et modifications complexes.

Notre utilisateur doit pouvoir interagir avec ce système avec l’invite commande élaboré suivant les opérations décrites ci-dessus.

2.2 CAHIER DE CHARGE

Objectif Principal : Création d’un émulateur de système de gestion de fichier inspiré du sgf présent dans LINUX.

Nous décrivons dans ce qui suit nos estimations (d'après le cahier des charges fournis par nos encadrants) de sous objectif à réaliser afin de réaliser l'objectif principal:

1. La création d'un fichier qui jouera le rôle d'un disque dur virtuel
2. La création d'un mini shell qui sera l'interface avec laquelle l'utilisateur exploite le système de gestion de fichier envisagé , ceci dit ;
    a) Notre mini shell doit être capable de détecter les commandes fournis par l'utilisateur
    b) La séparation des options et des paramètres
    c) La capacité de ce mini shell de distinguer les commandes correctes des commandes erronés
3. Mise en place d'une stratégie d'un système de gestion de fichier qui comprend ;
    a) La structuration d'un bloc de donnée et un inode
    b) La gestion de ces blocs et ses inodes dans un disque dur (creation , modification , suppression)
4. Exploiter le langage C afin d'implémenter cette stratégie ;
    a) Chercher les fonctions et méthodologies qui permettent cette implémentation
    b) Acquérir la capacité de diviser et exploiter le disque dur virtuel suivant nos besoins
    c) Utiliser ces acquis afin de définir des primitives qui interagissent directement avec le disque dur . Elaboration partielle de notre implémentation de l'SGF
    d) Tester statiquement ces primitives et finalisation de la creation de notre SGF
5. Effectuer le lien entre les commandes utilisateur et les primitives créés
    => encapsulation des fonctions inutiles pour l'utilisateur (Rendre le système "User friendly") , rendre ce système une **boîte noire .**

Stratégie d'implémentation

2.3 REPARTITION DES TACHES

Taches	Chargé de la tâche
Iheb BESBES	-Recherche individuelle sur les notion d'implémentation d'un système de gestion de fichiers -Mise en place de la stratégie du système de gestion de fichier (en trinôme) -implémentation du sgf (primitives et fonctions) (en binôme)
Mohamed Ilyess EL AJROUD	-Recherche individuelle sur les notion d'implémentation d'un système de gestion de fichiers -Mise en place de la stratégie du système de gestion de fichier (en trinôme) -test des primitives et fonction implémentés(en binôme)
Mohamed Firas SOURI	-Recherche individuelle sur les notion d'implémentation d'un système de gestion de fichiers -Mise en place de la stratégie du système de gestion de fichier (en trinôme) -liaison entre les primitives et commandes utilisateurs(en binôme)
Iyed CHEBERLI	-Recherche individuelle sur les notion d'implémentation d'un système de gestion de fichiers -La creation d'un mini shell (en binome)

	-test des primitives et fonction implémentés(en binôme)
Sami FAKHFAKH	-Recherche individuelle sur les notion d'implémentation d'un système de gestion de fichiers -La creation d'un mini shell (en binome) -liaison entre les primitives et commandes utilisateurs(en binôme)
Seifeddin BENJEMAA	-Recherche individuelle sur les notion d'implémentation d'un système de gestion de fichiers -Création du disque dur virtuel -implémentation du sgf (primitives et fonctions) (en binôme)

3 Réalisation

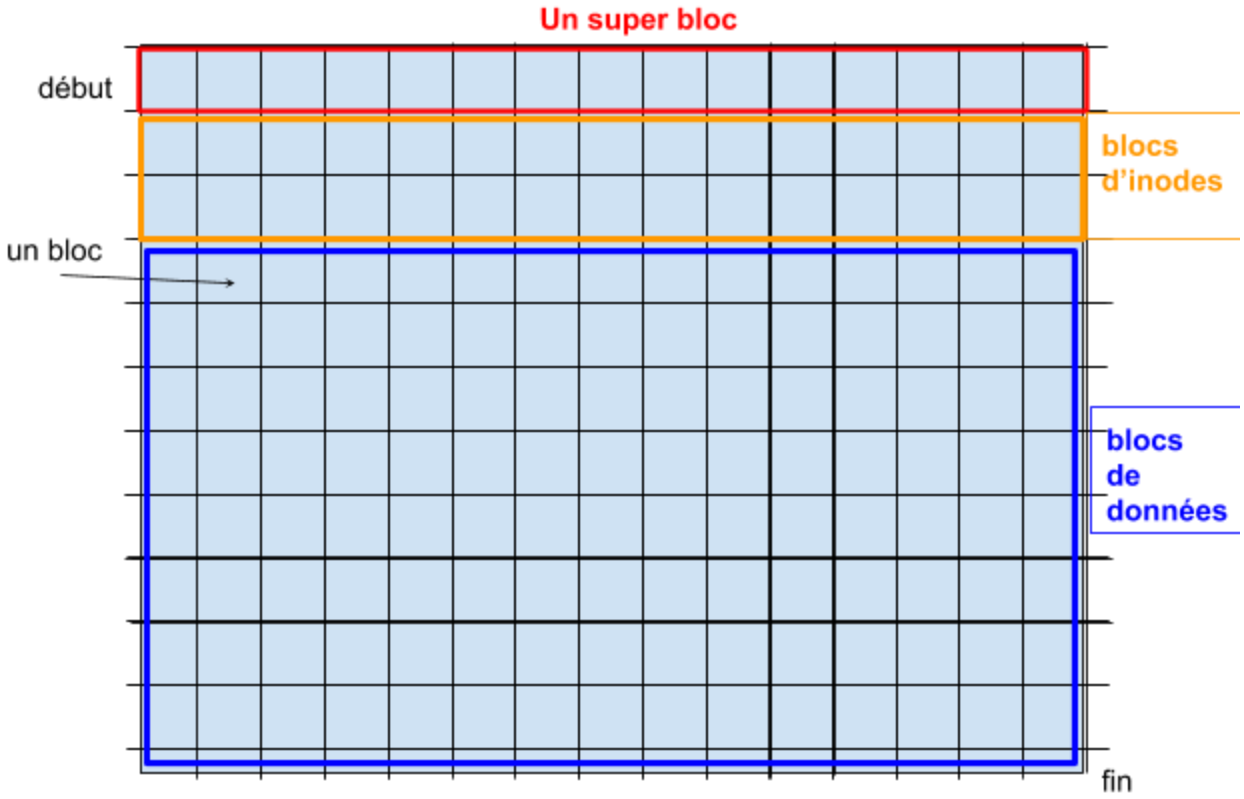
3.1 STRATEGIE D'IMPLEMENTATION :

Notre stratégie d'implémentation et prévision de structures de données:
Nous avons choisis une stratégie d'implémentation de système de gestion de fichiers trés proche de celle de LINUX mais plus simplifié qui s'oriente principalement autour du cahier des charges élaboré , notre disque dur sera divisé comme suit :

-Les blocs d'inodes : Ces blocs stockent les informations vitaux d'un fichier ou répertoire .On peut en citer le type du fichier, les droits d'accès , ...
Ces inodes contiendront aussi une table de pointeurs qui pointent sur les blocs de données . on a choisis de créer un seul pointeur indirect pour simplifier l'implémentation de ces inodes
-Un super bloc :Ce bloc contiendra les informations sur la totalité du disque tel que le nombre total de blocs , le nombre de blocs vides le nombre d'inodes vides ..
Remarque : On dit toujours UN super bloc parce qu'il n'existe qu'un seul qui caractérise le disque dur mais physiquement il est formé d'un ensemble de blocs .

-Des blocs de données : Ces blocs constituent chaque fichier de notre SGF.Un bloc est présenté sous forme d'un espace alloué sur disque dur qui permettra de stocker les données d'une partie ou d'une totalité d'un fichier .Les blocs de données sont alloués au fur et à mesure de l'extension du fichier .
Nous prévoyons aussi les constantes qui suivent :

- La taille d'un bloc qui sera 1024 octets
- Le nombre de blocs inodes qui sera 2500
- Le nombre restant des blocs qui sera 99900
- Ces nombres sont calculés d'après la taille de notre disque dur qui est 104 857 600 octets (environs 100 Mo)



Représentation simplifié du disque dur virtuel

3.2 STRUCTURE DES BLOCS

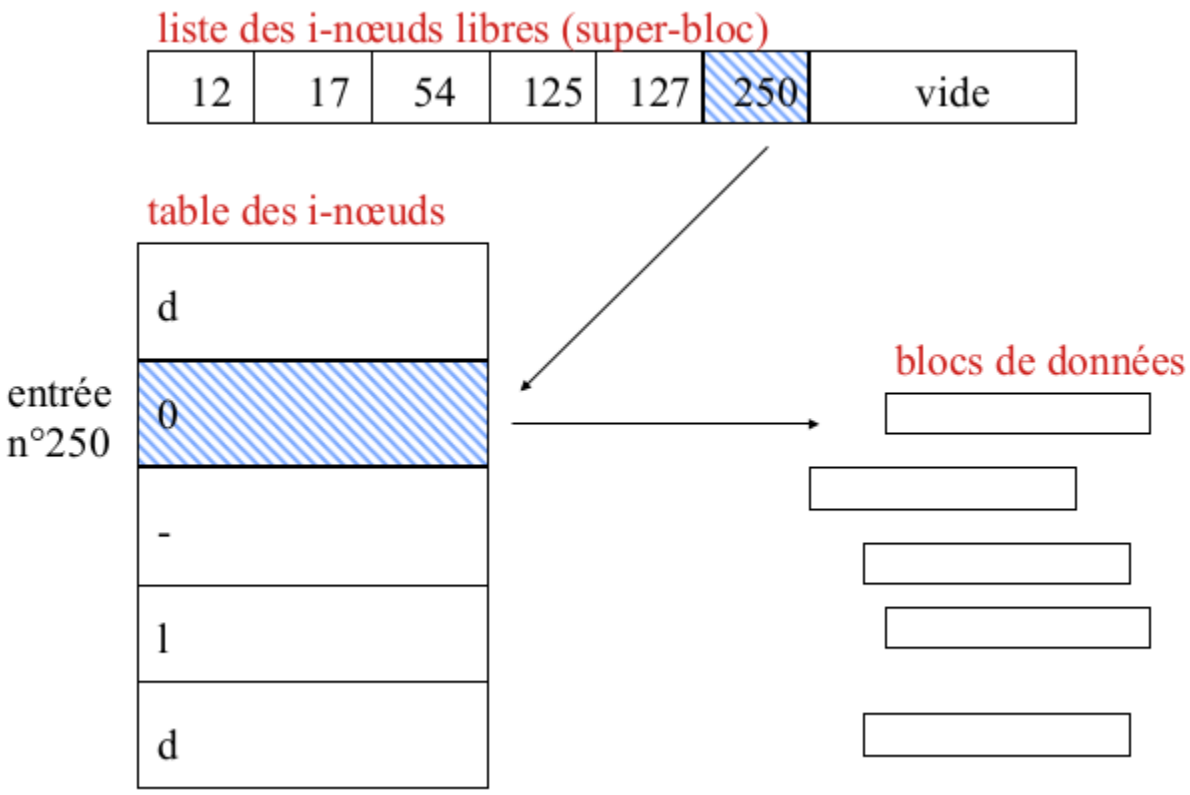
3.2.1 STRUCTURE d'un super-block

```
struct super_block
{
    int magic_number;
    int block_num;
    int inode_num;
    int free_blk;
    int free_inode;
};
```

Un super bloc contient les informations généraux du disque tel que l'espace disponible allouable ....ces informations sont stockées dans les variables internes du super bloc :
magic_number : un entier souvent trouvé dans le début ou près du début d'un fichier, il indique le format du fichier, il est appelé aussi "file signature" , nous utiliserons ce numéro pour caractériser notre disque dur et vérifier si un disque a été formaté selon notre stratégie ou pas.

- block_num: un entier qui represente le nombre de blocs totales du disque dur
- inode_num: représenté par un entier, il indique le nombre total d'inodes dans le disque
- free_blk: un entier qui représente le nombre de blocs libres dans notre disque dur

- free_inodes: le nombre d'inodes libre dans le système , type; entier.



Exemple d'utilisation d'un super bloc pour la création d'un nouveau fichier

3.2.2 STRUCTURE d'un fichier

Dans notre système , un fichier est constitué d'un inode qui contient toute informations nécessaire à la manipulation de ce fichier . Cet inode pointra sur un ou plusieurs blocs de données dans le disque dur .

Structure d'un bloc d'inode:

```
struct inode {
    int type;
    int num;
    int size;
    int uid;
    int gid;
    char mode[11];
    char name[MAX_FILE_NAME_LENGTH];
    int blocks[10];
```

```
                    int ind_blocks[30];
    char unused[15]; };
```

Cette structure représente les champs à implémenter dans un bloc d'inode , un inode est caractérisé par les champs qui suivent:

- Type : Un entier qui peut prendre 1 pour dire que c'est un inode de fichier ou l'objet pointé par cet inode est un fichier et 2 pour dire que c'est un inode de répertoire
- Num : est le numéro de l'inode dans un disque dur , c'est un numéro unique qui caractérise l'inode en utilisant ce numéro nous pouvons trouver l'inode.
- Size : Un entier qui décrit la taille du fichier
- Uid et gid : l'id de l'utilisateur et du groupe (on a introduit ces valeur pour faciliter l'implémentation de la notion d'utilisateur dans notre projet si possible)
- Mode[11]: Une chaîne de caractère qui contiendra les droit d'accès au fichier de l'inode.

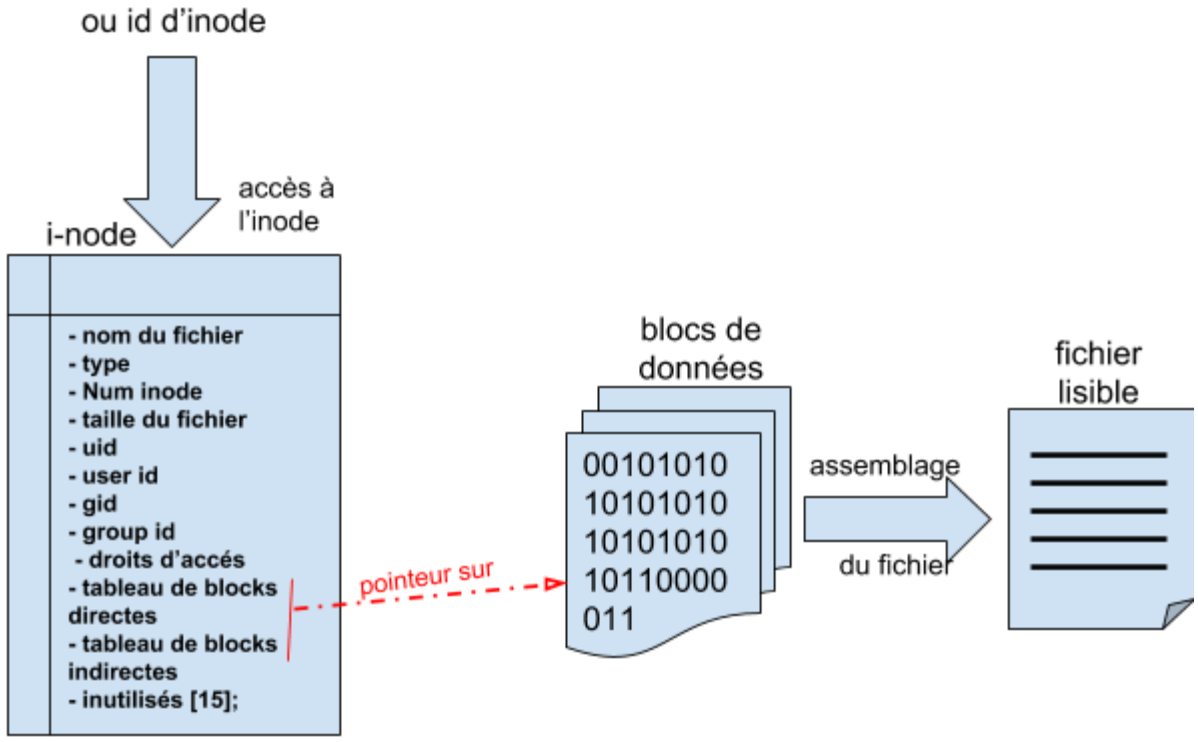
Name: Une chaîne de caractère qui contiendra le nom du fichier, normalement le nom du fichier n'est pas stocké dans l'inode, on a choisit de le mettre dans l'inode afin de simplifier l'implémentation.

- Blocks[10]: Un tableau d'entier qui contiendra les pointeur vers les blocs de données (ou sera stocké le contenu de notre fichier)
- Ind_blocks[30]: Un tableau de pointeurs indirectes simple (rajout de double et triple direction une fois le système est fonctionnel si possible).
- Unused[15]: un tableau non utilisé , nous voulons que la taille d'un inode soit égale à 256 octets

2.2 un bloc de donnée:

La taille d'un bloc sera de 1024 octets

Accès d'un fichier à partir de son nom



Chemin d'accés d'un fichier à partir de son nom

3.2.3 STRUCTURE d'un répertoire

Un répertoire est un ensemble de fichiers ou sous répertoires groupés en un seul endroit
Cette définition est représenté par la structure suivante :

```
struct dir
{
    struct dirEntry dentry[32];
};
```

Un répertoire est constitué de 32 DIREntry qui peuvent être soit des fichiers soit des sous-répertoires .

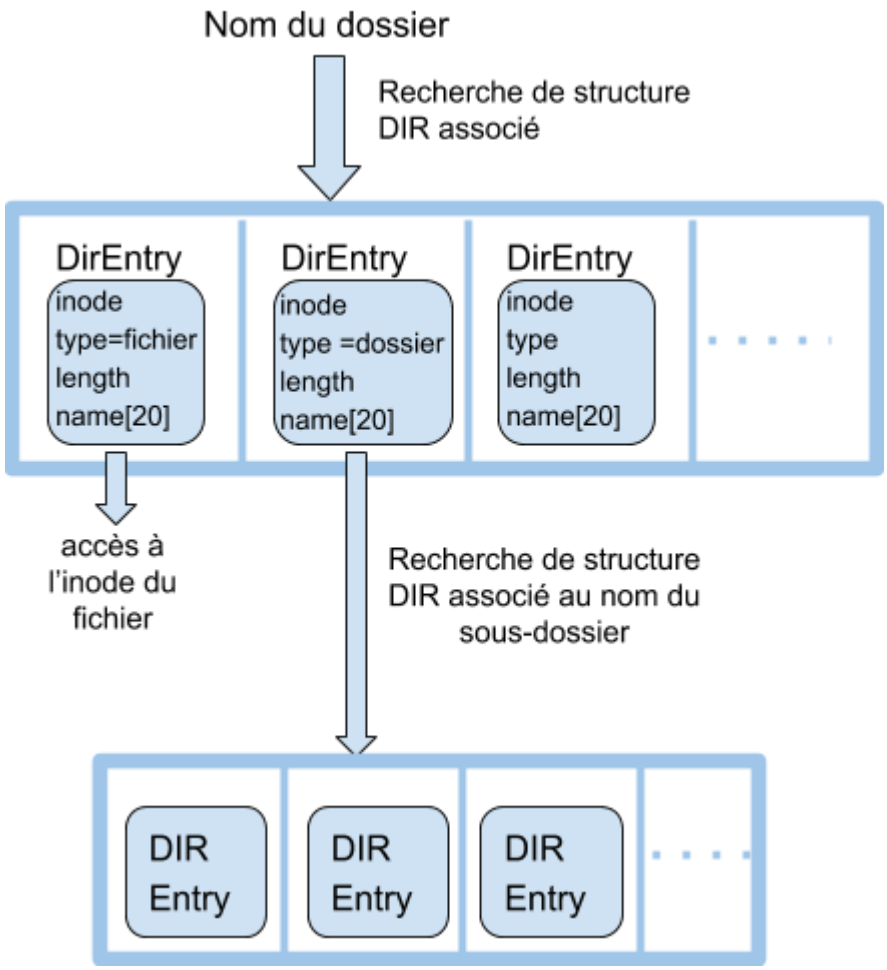
Structure d'un dirEntry :

```
struct dirEntry
{
    int inode;
    int type;
    int length;
    char name[20];
};
```

Cette structure permet de représenter le contenu d'un dossier dans notre SGF
dirEntry peut etre associer à un inode d'un fichier ou un inode d'un sous-dossier .

Cette structure va nous faciliter ultérieurement le déplacement entre dossiers par la création des liaisons entre les différents fichiers et la génération de liens symboliques .
Cette structure contiendra;

- le numéro de l'inode (entier),
- le type du fichier (entier) (fichier ou dossier)
- Length: qui représente la longueur du nom du fichier (entier) .
- Name: nom du fichier (chaîne de caractère).



Accès d'un dossier et ces sous-dossiers à partir de son nom

3.3 FONCTIONS BASIQUES

Les fonctions basiques sont les piliers de notre système de fichiers.Ils permettent principalement la manipulation (lecture,écriture,modification,suppression) de tout blocs qui constituent le disque dur .

- **int read_block(int number, void * block)**

Void * permet de passer plusieurs types de structures différentes à cette fonction en entrée .ceci lui permettra de prendre n'importe quel type de bloc écrit dans le disque dur virtuel et d'en extraire les données stockés qui peuvent être des inodes , un super bloc ou des blocs de données

- **int write_block(int number, void * block)**

Cette fonction permet d'insérer un bloc donné dans notre disque dur elle prend en paramètres un entier qui désigne le numéro de ce blocs dans la carte des blocs du disque et un void * qui désigne le bloc à écrire l'entier renvoyé permet de déterminer si la fonction a été exécuté avec succès ou pas

- **int write_inode(int number , void * inode)**

Similaire aux fonctions précédents cette fonctions permet d'ajouter un inode suivant un numéro donné dans la carte d'inode du disque dur l'entier renvoyé permet aussi de voir si la fonction a été exécuté avec succès ou pas .

- **int read_inode(int number , void * inode)**

Cette fonction permet de récupérer un inode dans void inode suivant le numéro d'inode donné, le type de retour permet de savoir si la fonction a été exécuté correctement ou pas

- **void format_disk ()**

Comme son nom l'indique cette fonction permet de formater le disque dur suivant nos besoins mentionnés au par avant dans ce rapport, elle initialise le disque dur, calcule le nombre de blocs présents dans ce disque, initialise et inséré un super bloc, et puis crée le répertoire root qui représente la racine de notre arborescence de fichiers elle réutilise les write_inode et write_block pour la création de ce répertoire
Cette fonction permet également de vérifier si le répertoire a été déjà formaté en testant la présence du magic_number et en le comparant avec notre numéro magique associé.

Ces fonction seront implémentés avec les primitives décrites dans le cahier de charge fournis par nos encadrants pour atteindre notre objectif de création du sgf, dans une étape finale nous procéderons à lier ces primitives et fonctions avec les commandes de notre utilisateur.