Design Document for Partition Management

Overview

This document explains the design choices made for the partition management system in the project. It provides a detailed justification for the structures used to represent the partition, inodes, blocks, directories, and the overall file system.

---

1. **Structure `Block**

Definition

```
typedef struct Block {
    uint32_t index;       ///< Index of the block in the partition
    uint8_t *data;        ///< Pointer to the block's data
    bool is_free;         ///< Indicates whether the block is free or
used
} Block;
```

- **Justification** :
  - Un bloc est une unité de stockage de base dans un système de fichiers.
  - La structure `Block` permet de représenter chaque bloc avec un index unique, un pointeur vers ses données, et un indicateur de disponibilité (`is_free`).
  - Cela facilite la gestion des blocs libres et utilisés via un bitmap, tout en permettant un accès direct aux données stockées dans chaque bloc.

2. **Structure `Inode`**

Robin

Definition

```
typedef struct Inode {
    uint32_t id;              ///< Unique identifier for the inode
    uint32_t size;            ///< Size of the file (in bytes)
    uint32_t blocks[12];      ///< Direct pointers to data blocks
    uint32_t indirect_block;  ///< Pointer to an indirect block (if
needed)
    uint32_t double_indirect; ///< Pointer to a double-indirect block
(if needed)
    uint16_t permissions;     ///< File permissions (UNIX style:
rwxrwxrwx)
    uint16_t links_count;     ///< Number of links pointing to this
inode
    uint32_t owner_id;        ///< Owner ID
    uint32_t group_id;        ///< Group ID
```

```
    uint64_t created_at;       ///< Creation timestamp
    uint64_t modified_at;      ///< Last modification timestamp
    uint64_t accessed_at;      ///< Last access timestamp
    bool is_directory;         ///< Indicates whether the inode
represents a directory
} Inode;
```

- **Justification** :
  - Les inodes sont essentiels pour stocker les métadonnées des fichiers et répertoires (taille, permissions, timestamps, etc.).
  - Les pointeurs directs et indirects permettent de gérer efficacement les fichiers de différentes tailles, en optimisant l'accès aux données.
  - L'utilisation de champs comme `permissions`, `owner_id`, et `group_id` garantit une compatibilité avec les systèmes de fichiers de type UNIX.
  - Le champ `is_directory` permet de différencier les fichiers des répertoires, simplifiant ainsi les opérations spécifiques aux répertoires.

3. Structure `Directory`

Definition

```
typedef struct Directory {
    uint32_t parent_inode;    ///< Inode of the parent directory
    uint32_t entries[128];    ///< Inodes of files or subdirectories
    char names[128][256];     ///< Names of files or subdirectories
    uint32_t entry_count;     ///< Number of entries in the directory
} Directory;
```

- **Justification** :
  - Les répertoires sont représentés comme des conteneurs d'entrées, chaque entrée associant un nom de fichier ou de sous-répertoire à un inode.
  - La structure `Directory` inclut un tableau pour stocker les indices des inodes et un tableau pour les noms correspondants, ce qui permet une gestion simple et rapide des entrées.
  - Le champ `parent_inode` facilite la navigation dans la hiérarchie des répertoires.

Robin

4. Structure `Superblock`

Definition

```
typedef struct Superblock {
    char magic[8];            ///< Unique identifier for the file system
    uint32_t total_size;      ///< Total size of the partition
    uint32_t block_size;      ///< Size of a block
    uint32_t total_blocks;    ///< Total number of blocks
    uint32_t free_blocks;     ///< Number of free blocks
    uint3232 file_table_start;///< Start of the file table (not used
```

```
here)
    uint32_t data_start;      ///< Start of the data blocks
} Superblock;
```

- **Justification** :
  - Le superblock contient des métadonnées globales sur la partition, telles que la taille totale, la taille des blocs, et le nombre de blocs libres.
  - Ces informations sont essentielles pour initialiser et gérer le système de fichiers, en fournissant un point d'entrée centralisé pour les opérations.
  - Le champ `magic` permet d'identifier le système de fichiers, garantissant ainsi la compatibilité et la validation des partitions.

5. Structure `Partition`

Definition

```
typedef struct Partition {
    uint32_t total_size;      ///< Total size of the partition (in
bytes)
    uint32_t block_size;      ///< Size of a block (in bytes)
    uint3232 total_blocks;    ///< Total number of blocks
    uint32_t free_blocks;     ///< Number of free blocks
    uint8_t *bitmap;          ///< Bitmap to track used and free blocks
    Block *blocks;            ///< Array of blocks
    void *data;               ///< Pointer to the raw partition data
} Partition;
```

- **Justification** :
  - La structure `Partition` regroupe les informations sur la partition brute, y compris le bitmap pour suivre l'état des blocs et un tableau de blocs.
  - Cela permet une gestion centralisée des blocs de données, tout en offrant une abstraction claire entre les données brutes et les structures de haut niveau comme les inodes.

Robin

6. Structure `FileSystem`

Definition

```
typedef struct FileSystem {
    Superblock superblock;    ///< Superblock of the partition
    Inode *inode_table;       ///< Table of inodes
    Partition partition;      ///< Raw partition
} FileSystem;
```

- **Justification** :

- La structure `FileSystem` combine toutes les composantes du système de fichiers (superblock, table des inodes, partition) en une seule entité.
- Cela simplifie la gestion globale du système de fichiers, en permettant de passer un seul pointeur pour accéder à toutes les structures nécessaires.

7. **Bitmap pour la gestion des blocs**

- **Justification** :
  - Le bitmap est une méthode efficace pour suivre l'état des blocs (libres ou utilisés) dans la partition.
  - Il réduit la mémoire nécessaire par rapport à une liste chaînée ou une autre structure, tout en permettant des opérations rapides pour trouver ou libérer des blocs.

8. **Table des inodes**

- **Justification** :
  - La table des inodes est un tableau fixe qui permet un accès rapide aux métadonnées des fichiers et répertoires.
  - L'utilisation d'un tableau fixe simplifie la gestion des inodes, bien que cela impose une limite maximale au nombre de fichiers.

9. **Choix des types de données**

- **Justification** :
  - Les types comme `uint32_t` et `uint8_t` garantissent une portabilité et une précision dans la gestion des tailles et indices.
  - Les timestamps utilisent `uint64_t` pour représenter les dates et heures avec une grande précision, compatible avec les systèmes modernes.

Conclusion

Les structures choisies dans les fichiers `partition.h` et `partition.c` sont conçues pour offrir un équilibre entre simplicité, efficacité, et compatibilité avec les systèmes de fichiers traditionnels. Elles permettent une gestion modulaire et extensible des partitions, tout en respectant les contraintes de performance et de mémoire.

Robin