MP7: Simple File System

The Simple File System provides a simple file management solution built on an inode-based design. It supports fundamental operations such as file creation, deletion, sequential read/write, and disk mounting. This document details the implementation of the `File` and `FileSystem` classes, structured across `file.C/H` and `file_system.C/H` files.

file.H

The file.H header file defines the File class, encapsulating file-related operations and maintaining metadata for sequential read/write functionality.

File Class: The File class enables interaction with files stored on a disk. It maintains the current read/write position and a cached block for efficient I/O.

Private Members:

- **inode**: Pointer to the file's inode containing metadata (file size, storage block number).
- **fs**: Reference to the associated FileSystem instance.
- **current_position**: Tracks the current position in the file for read/write operations.
- **block_cache**: Cache for a single block of file data.

Public Members:

- **Constructor: File(FileSystem* _fs, int _id)**
  - Initializes the file handle, associates it with an inode, and loads the block into the cache.
- **Destructor: ~File()**
  - Flushes cached data to the disk and updates the inode list.
- **int Read(unsigned int _n, char* _buf)**
  - Reads _n bytes from the current position into _buf without exceeding file size.
- **int Write(unsigned int _n, const char* _buf)**
  - Writes _n bytes from _buf to the current position, extending file size if necessary.
- **void Reset()**
  - Resets the current position to the start of the file.
- **bool EoF()**
  - Checks if the current position is at the end of the file

file.C

The file.C implementation provides the logic for file operations, focusing on sequential access and efficient caching.

Class Implementation

Constructor: Initializes the file by locating its inode and loading the associated block into the cache. Throws an error if the file does not exist.

Destructor: Writes cached changes to the disk and updates the inode metadata before releasing resources.

int Read(unsigned int _n, char* _buf)

- Calculates the maximum readable bytes to avoid reading beyond file size.
- Copies data from the cached block to _buf and advances current_position.
- Returns the number of bytes read.

int Write(unsigned int _n, const char* _buf)

- Calculates the maximum writable bytes to avoid exceeding the block size.
- Copies data from _buf to the cached block and advances current_position.
- Updates file size and flushes changes to the disk.
- Returns the number of bytes written.

void Reset()

- Sets current_position to 0.

bool EoF()

- Returns true if current_position is greater than or equal to file size.

file_system.H

The file_system.H header file defines the FileSystem class, responsible for managing inodes, free blocks, and file operations.

FileSystem Class: The FileSystem class organizes files using an inode list and a free block list stored on disk. It provides methods for file creation, deletion, and lookup.

Private Members:

- **disk: Pointer to the associated disk instance.**
- **inodes: Array of inodes representing files in the system.**
- **free_blocks: Bitmap tracking free and used blocks.**
- **size: Total size of the file system in blocks.**

Public Members:

- **Constructor: FileSystem()**
  - Initializes local data structures but does not associate a disk.
- **Destructor: ~FileSystem()**
  - Saves the inode and free block lists to the disk before cleanup.
- **bool Mount(SimpleDisk* _disk)**
  - Reads inode and free block lists from the disk, associating them with the file system.
- **static bool Format(SimpleDisk* _disk, unsigned int _size)**
  - Wipes the disk and initializes a new file system, marking system-reserved blocks as used.
- **Inode* LookupFile(int _file_id)**
  - Searches the inode list for a file with the given ID.
- **bool CreateFile(int _file_id)**
  - Allocates an inode and a block for a new file.
- **bool DeleteFile(int _file_id)**
  - Frees a file's block and marks its inode as unused.

Inode Class: The Inode class represents metadata for a file, including its ID, size, and storage block.

Public Members:

- **Constructors:** Default and parameterized constructors for easy initialization.
- **Accessors/Mutators:**
  - **getId()**: Returns ID of the file.
  - **getSize()**: Returns size of the file.
  - **getBlockNumber()**: Returns block number of the file.
  - **setSize(unsigned int _size)**: Sets size of the file.
  - **setBlockNumber(unsigned int _block_number)**: Sets the Block number of the file.

file.C

The file_system.C implementation handles the core logic of file system operations, focusing on inode management and disk interaction.

Class Implementation

Constructor: Allocates memory for the inode list and free block list.

Destructor: Saves the inode and free block lists to the disk and releases allocated memory.

bool Mount(SimpleDisk* _disk)

- Reads the inode list and free block list from the disk into memory.
- Returns true on success.

static bool Format(SimpleDisk* _disk, unsigned int _size)

- Initializes an empty inode list and free block list, writing them to the disk..

bool CreateFile(int _file_id)

- Checks if the file already exists, allocates an unused inode, and assigns a free block.

bool DeleteFile(int _file_id)

- Frees the block occupied by the file and marks its inode as unused.

Utility Methods

void writeBlock(unsigned int block_number, unsigned char* buffer)

- Writes a block of data to the disk.

void readBlock(unsigned int block_number, unsigned char* buffer)

- Reads a block of data from the disk.

void updateInode(Inode* inode)

- Saves the inode list to the disk.

Debugging

- Debugging print statements are used for:
- File System Operations: Logs file creation, deletion, mounting, and formatting.
- Disk Interaction: Logs block read/write operations.
- Error Handling: Logs errors like missing files or insufficient resources