Virtual File System Using Inode Data Structure

This project implements a virtual file system in Python using an inode data structure, emulating disk operations within a single file.

Project Structure

```
virtual_file_system/
├── superblock.py
├── inode.py
├── virtual_disk.py
├── file_system.py
├── cli_commands.py
├── main.py
└── README.md
```

Features

- **Disk Emulation**: Uses a single file (`virtual_disk.vdsk`) to emulate disk operations.
- **Inode Structure**: Implements inodes with direct pointers, single indirect, and double indirect pointers.

- **Basic File Operations**: Supports creating, opening, closing, reading, writing, and deleting files.
- **Command-Line Interface**: Interact with the virtual file system through CLI commands.

How to Run

1. **Clone the Repository**

```
git clone https://github.com/yourusername/virtual_file_system.git
cd virtual_file_system
```

2. **Run the Program**

```
python main.py
```

   Ensure you have Python 3 installed.

Available Commands

- `create_disk`: Create a new virtual disk.
- `mount_disk`: Mount the virtual disk.
- `unmount_disk`: Unmount the virtual disk.
- `create_file`: Create a new file.
- `open_file`: Open an existing file.
- `close_file`: Close an open file descriptor.
- `write_file`: Write data to an open file.
- `read_file`: Read data from an open file.
- `delete_file`: Delete a file.
- `exit`: Exit the program.

Usage Example

1. **Create a Disk**

```
Enter command: create_disk
Disk created successfully.
```

2. **Mount the Disk**

```
Enter command: mount_disk
Enter username: user1
Enter password:
Disk mounted successfully.
```

3. **Create a File**

```
Enter command: create_file
Enter filename (use numbers for simplicity): 1
File '1' created with inode 1.
```

4. **Open the File**

```
Enter command: open_file
Enter filename (inode number): 1
Enter mode ('r' for read, 'w' for write): w
File '1' opened with file descriptor 0.
File descriptor: 0
```

5. **Write to the File**

```
Enter command: write_file
Enter file descriptor: 0
Enter data to write: Hello, World!
Wrote data to file descriptor 0.
```

6. **Read from the File**

```
Enter command: read_file
Enter file descriptor: 0
Read data from file descriptor 0.
Data read:
Hello, World!
```

7. **Close the File**

   ```
   Enter command: close_file
   Enter file descriptor: 0
   File descriptor 0 closed.
   ```

8. **Delete the File**

   ```
   Enter command: delete_file
   Enter filename (inode number): 1
   File '1' deleted.
   ```

9. **Unmount the Disk**

   ```
   Enter command: unmount_disk
   Disk unmounted successfully.
   ```

10. **Exit the Program**

    ```
    Enter command: exit
    ```

Implementation Details

- **Inode Data Structures**: Each inode stores file metadata and pointers to data blocks.
- **Disk Emulation**: The virtual disk is a file that contains the superblock, inodes, and data blocks.
- **File Operations**: Basic operations are implemented to interact with files in the virtual file system.
- **Security Considerations**: User authentication is prompted when mounting the disk (password validation not implemented).

Extending the Project

- **Directory Structure**: Implement a hierarchical directory system.
- **User Authentication**: Add password validation and user management.
- **Permissions**: Enforce file permissions based on user access rights.
- **Indirect Pointers**: Fully implement single and double indirect pointers.
- **Error Handling**: Improve input validation and error messages.
- **Encryption**: Integrate file encryption for security.

License

Contributions

Contributions are welcome! Please open issues or submit pull requests on GitHub.