

**Department of Computer Engineering**

**Course:** Systems Programming Lab

**Course Code:** BTECCE21507

**Lab Mini-Project -Report**

**Guidance By -** Prof. Swati. S

**Topic:** File System Implementation

**By**

|  |  |  |
| --- | --- | --- |
| **Roll No** | **SRN** | **Name of Student** |
| 38 | 202200930 | Shyamal S. Patil |

**Project Description:**

**1. File System Structure:**

* Create a hierarchical file system structure, consisting of directories and files.
* Implement a root directory as the starting point.

**2. File and Directory Management:**

* Create, delete, move, and rename files and directories.
* List the contents of directories.
* Support for nested directories.

**3. File Operations:**

* Read and write data to files.
* Handle file permissions and access control.

**4. Metadata:**

* Store metadata for files and directories (e.g., creation date, modification date, size, owner, permissions).

**5. Error Handling:**

* Implement robust error handling for common file system operations.

**6. Disk Space Management:**

* Track available disk space.
* Implement mechanisms for handling disk space allocation and deallocation.

**7. Security and Access Control:**

* Implement user authentication and authorization.
* Enforce access control lists (ACLs) for files and directories.

**8. Disk Persistence:**

* Implement mechanisms for storing and retrieving the file system structure on disk.

**9. File System Commands:**

* Create a command-line interface or GUI for users to interact with the file system.
* Implement a set of commands for file and directory management (e.g., **ls**, **mkdir**, **touch**, **mv**, **rm**, etc.).

**10. Multi-User Support:**

* Support multiple users with their own home directories and permissions.
* Implement user account management.

**11. Advanced Features:**

* Support for symbolic links.
* Implement search and indexing capabilities.
* Implement file versioning and snapshots.

**12. Testing:**

* Develop a comprehensive test suite to validate the correctness of file system operations.

**13. Documentation:**

* Create detailed documentation and user guides for using the file system.

**14. Performance Optimization:**

* Optimize performance for common file operations (e.g., caching, efficient data structures).

**15. Compatibility:**

* Ensure compatibility with standard file system APIs and file formats (e.g., POSIX).

**16. Backup and Recovery:**

* Implement backup and recovery mechanisms to protect against data loss.

**Algorithm:**

**Step 1: Data Structures**

* Define data structures to represent files, directories, and metadata. You can use classes or data structures for this purpose.

**Step 2: Directory Management**

* Create a root directory as the starting point of the file system.
* Implement operations for creating, deleting, renaming, and listing directories.

**Step 3: File Operations**

* Implement file creation, deletion, reading, and writing.
* Develop algorithms for reading and writing data blocks from and to the disk.

**Step 4: Metadata Management**

* Store metadata for files and directories, including file size, timestamps, permissions, and owner information.
* Implement data structures and algorithms for managing metadata.

**Step 5: Disk Space Management**

* Develop an algorithm to track available and used disk space.
* Allocate and deallocate disk blocks for files.

**Step 6: Security and Access Control**

* Implement access control mechanisms like access control lists (ACLs) or permission bits.
* Develop algorithms to check permissions and access rights.

**Step 7: Error Handling**

* Implement algorithms to handle and report errors during file operations. Common errors include insufficient disk space, file not found, or permission denied.

**Step 8: Disk Persistence**

* Implement algorithms to save and load the file system structure to/from persistent storage, like a hard drive or SSD.
* Ensure data integrity and consistency during these operations.

**Step 9: User Authentication and Authorization**

* Develop algorithms for user authentication and authorization.
* Check user privileges and access rights during file system operations.

**Step 10: Command-Line Interface (CLI) or GUI**

* Create a user interface for interacting with the file system.
* Develop parsers and command execution algorithms for handling user commands.

**Step 11: File Search and Indexing**

* Implement search algorithms for efficient file lookup, such as binary search or indexing structures.

**Step 12: Backup and Recovery**

* Develop backup algorithms to periodically save the file system state.
* Implement recovery algorithms to restore the file system to a consistent state after a crash.

**Step 13: Performance Optimization**

* Optimize algorithms for common file operations to enhance performance.
* Implement caching mechanisms for frequently accessed data.

**Step 14: Compatibility**

* Ensure your file system is compatible with standard file system APIs and adheres to relevant file system standards, such as POSIX.

**Code Implementation:**

import java.util.ArrayList;

import java.util.List;

class File {

private String name;

private String content;

public File(String name) {

this.name = name;

this.content = "";

}

public String getName() {

return name;

}

public String getContent() {

return content;

}

public void setContent(String content) {

this.content = content;

}

}

class Directory {

private String name;

private List<File> files;

private List<Directory> subDirectories;

public Directory(String name) {

this.name = name;

this.files = new ArrayList<>();

this.subDirectories = new ArrayList<>();

}

public String getName() {

return name;

}

public List<File> getFiles() {

return files;

}

public List<Directory> getSubDirectories() {

return subDirectories;

}

public void addFile(File file) {

files.add(file);

}

public void createSubDirectory(String name) {

Directory newDir = new Directory(name);

subDirectories.add(newDir);

}

public Directory getSubDirectory(String name) {

for (Directory dir : subDirectories) {

if (dir.getName().equals(name)) {

return dir;

}

}

return null;

}

public File getFile(String name) {

for (File file : files) {

if (file.getName().equals(name)) {

return file;

}

}

return null;

}

}

public class SimpleFileSystem {

private Directory root;

public SimpleFileSystem() {

root = new Directory("/");

}

public Directory getRoot() {

return root;

}

public static void main(String[] args) {

SimpleFileSystem fileSystem = new SimpleFileSystem();

Directory root = fileSystem.getRoot();

// Create a file and add content

File file1 = new File("file1.txt");

file1.setContent("Hello, World!");

root.addFile(file1);

// Create a subdirectory and add a file inside it

root.createSubDirectory("subdir1");

Directory subdir1 = root.getSubDirectory("subdir1");

File file2 = new File("file2.txt");

file2.setContent("This is a subdirectory.");

subdir1.addFile(file2);

// Retrieve and print the content of files

System.out.println(root.getFile("file1.txt").getContent());

System.out.println(subdir1.getFile("file2.txt").getContent());

}

}

**Output Screenshot:**

