advanced-filesystem-assignment.md 2024-12-19

Advanced File System Implementation - Phase 1

Building upon your basic file system implementation, enhance it with advanced features and robust error handling.

1. Advanced Operations

Core Functionality Extensions

class Node {

string name;

bool isDirectory;

string content;

Node\* firstChild;

Node\* nextSibling;

Node\* parent;

// New metadata fields

time\_t createdAt;

time\_t modifiedAt;

size\_t fileSize;

string owner;

unsigned int permissions; // Unix-style permissions

bool isSymLink;

string linkTarget; // For symbolic links

};

Required Operations

. File Management

mv(string source, string dest): Move files/directories

cp(string source, string dest): Copy files/directories

rename(string oldName, string newName): Rename files/directories

rmdir(string path): Remove directory and contents

ln -s(string target, string linkName): Create symbolic link

. Metadata Operations

stat(string path): Display file/directory metadata

chmod(string path, unsigned int mode): Change permissions

chown(string path, string newOwner): Change ownership

. Search Operations

find(string pattern): Search by name pattern

find -i(string pattern): Case-insensitive search

grep(string content): Search file contents

1 / 4

advanced-filesystem-assignment.md 2024-12-19

2. Persistence System

File System Serialization

class FileSystem {

// Existing methods...

void saveToFile(string filename); // Save current state void loadFromFile(string filename); // Restore state

// Helper methods

void serializeNode(Node\* node, ofstream& out);

Node\* deserializeNode(ifstream& in);

};

3. Command Line Interface

void startCLI() {

while (true) {

cout << pwd() << "> ";

string command;

getline(cin, command);

executeCommand(command);

}

}

4. Sample Usage

int main() {

FileSystem fs;

// Create and populate directories

fs.mkdir("/home/user/documents");

fs.touch("/home/user/documents/report.txt", "Annual Report");

// Copy and move operations

fs.cp("/home/user/documents/report.txt", "/home/user/backup/"); fs.mv("/home/user/documents", "/home/user/old\_docs");

// Permissions and ownership

fs.chmod("/home/user/backup/report.txt", 0644);

fs.chown("/home/user/backup/report.txt", "admin");

// Create symbolic link

fs.ln("-s", "/home/user/backup/report.txt", "/home/user/report\_link");

// Search files

2 / 4

advanced-filesystem-assignment.md 2024-12-19

fs.find("\*.txt");

// Save file system state

fs.saveToFile("filesystem\_backup.dat");

return 0;

}

5. Error Handling Requirements

. Operation Validation

Check permissions before operations

Validate paths and file existence

Prevent circular references in moves

Handle duplicate files

. Resource Management

Memory leak prevention

Proper cleanup during deletions

Handle large file operations

6. Implementation Guidelines

. Error Handling

class FileSystemException : public std::exception { string message;

public:

FileSystemException(string msg) : message(msg) {} const char\* what() const noexcept override {

return message.c\_str();

}

};

// Usage in operations

void mv(string source, string dest) {

if (!exists(source))

throw FileSystemException("Source path does not exist"); if (isCircularReference(source, dest))

throw FileSystemException("Circular reference detected"); // ... implementation

}

. Metadata Implementation

3 / 4

advanced-filesystem-assignment.md 2024-12-19

struct Metadata {

time\_t createdAt;

time\_t modifiedAt;

size\_t fileSize;

string owner;

unsigned int permissions;

string toString() const {

// Format metadata for display

}

};

Evaluation Criteria

. Implementation completeness of advanced operations

. Proper error handling and validation

. Memory management and resource cleanup

. CLI usability and robustness

. Code organization

This phase builds upon the basic file system to create a more realistic and robust implementation, emphasizing error handling, user interaction, and advanced file operations.

4 / 4