**Day 3: CPP code:**

#include <iostream>

#include <string>

#include <dirent.h>

#include <unistd.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <cstring>

#include <vector>

#include <fstream>

#include <sstream> // Include this for std::istringstream

class FileExplorer {

private:

std::string currentDirectory;

public:

FileExplorer() {

char buffer[1024];

if (getcwd(buffer, sizeof(buffer)) != nullptr) {

currentDirectory = std::string(buffer);

}

}

void start() {

std::string command;

while (true) {

std::cout << currentDirectory << " > ";

std::getline(std::cin, command);

if (command == "exit") break;

executeCommand(command);

}

}

private:

void executeCommand(const std::string& command) {

if (command == "list") {

listFiles();

} else if (command.substr(0, 3) == "cd ") {

changeDirectory(command.substr(3));

} else if (command.substr(0, 4) == "copy") {

auto args = parseCommand(command);

if (args.size() == 3) {

copyFile(args[1], args[2]);

}

} else if (command.substr(0, 4) == "move") {

auto args = parseCommand(command);

if (args.size() == 3) {

moveFile(args[1], args[2]);

}

} else if (command.substr(0, 6) == "delete") {

auto args = parseCommand(command);

if (args.size() == 2) {

deleteFile(args[1]);

}

} else if (command.substr(0, 6) == "create") {

auto args = parseCommand(command);

if (args.size() == 2) {

createFile(args[1]);

}

} else if (command.substr(0, 6) == "search") {

auto args = parseCommand(command);

if (args.size() == 2) {

searchFiles(currentDirectory, args[1]);

}

} else if (command.substr(0, 4) == "chmod") {

auto args = parseCommand(command);

if (args.size() == 3) {

setPermissions(args[1], args[2]);

}

} else {

std::cout << "Unknown command: " << command << std::endl;

}

}

void listFiles() {

DIR\* dir = opendir(currentDirectory.c\_str());

if (dir == nullptr) {

std::cerr << "Error opening directory!" << std::endl;

return;

}

struct dirent\* entry;

while ((entry = readdir(dir)) != nullptr) {

std::cout << entry->d\_name << std::endl;

}

closedir(dir);

}

void changeDirectory(const std::string& newDir) {

if (chdir(newDir.c\_str()) == 0) {

char buffer[1024];

if (getcwd(buffer, sizeof(buffer)) != nullptr) {

currentDirectory = std::string(buffer);

}

} else {

std::cerr << "Error: Cannot change directory to " << newDir << std::endl;

}

}

void copyFile(const std::string& source, const std::string& destination) {

std::ifstream src(source, std::ios::binary);

std::ofstream dest(destination, std::ios::binary);

dest << src.rdbuf();

src.close();

dest.close();

std::cout << "File copied to " << destination << std::endl;

}

void moveFile(const std::string& source, const std::string& destination) {

if (rename(source.c\_str(), destination.c\_str()) == 0) {

std::cout << "File moved to " << destination << std::endl;

} else {

std::cerr << "Error moving file!" << std::endl;

}

}

void deleteFile(const std::string& path) {

if (unlink(path.c\_str()) == 0) {

std::cout << "File deleted: " << path << std::endl;

} else {

std::cerr << "Error deleting file!" << std::endl;

}

}

void createFile(const std::string& fileName) {

std::ofstream file(fileName);

if (file) {

std::cout << "File created: " << fileName << std::endl;

} else {

std::cerr << "Error creating file!" << std::endl;

}

file.close();

}

void searchFiles(const std::string& directory, const std::string& pattern) {

DIR\* dir = opendir(directory.c\_str());

if (dir == nullptr) {

std::cerr << "Error opening directory!" << std::endl;

return;

}

struct dirent\* entry;

while ((entry = readdir(dir)) != nullptr) {

if (strstr(entry->d\_name, pattern.c\_str()) != nullptr) {

std::cout << "Found: " << entry->d\_name << std::endl;

}

if (entry->d\_type == DT\_DIR && strcmp(entry->d\_name, ".") != 0 && strcmp(entry->d\_name, "..") != 0) {

searchFiles(directory + "/" + entry->d\_name, pattern);

}

}

closedir(dir);

}

void setPermissions(const std::string& path, const std::string& mode) {

mode\_t perm = std::stoi(mode, 0, 8);

if (chmod(path.c\_str(), perm) == 0) {

std::cout << "Permissions set to " << mode << std::endl;

} else {

std::cerr << "Error setting permissions!" << std::endl;

}

}

std::vector<std::string> parseCommand(const std::string& command) {

std::vector<std::string> tokens;

std::string token;

std::istringstream tokenStream(command);

while (std::getline(tokenStream, token, ' ')) {

tokens.push\_back(token);

}

return tokens;

}

};

int main() {

FileExplorer explorer;

explorer.start();

return 0;

}

This C++ application is a basic file explorer that uses a command-line interface (CLI) to let users interact with the file system. Some of the fundamental file operations it offers are listing files, rearranging directories, copying, moving, deleting, creating, searching, and modifying file permissions.

**FileExplorer Class:**

**Data Member:**

std::string currentDirectory: Stores the current working directory path.

**Constructor:**

Initializes currentDirectory to the current working directory using getcwd.

**Methods:**

start(): Begins the command loop where the user can input commands until "exit" is entered.

executeCommand(const std::string& command): Parses and executes the user's command.

listFiles(): Lists all files and directories in the current directory.

changeDirectory(const std::string& newDir): Changes the current working directory.

copyFile(const std::string& source, const std::string& destination): Copies a file from source to destination.

moveFile(const std::string& source, const std::string& destination): Moves or renames a file.

deleteFile(const std::string& path): Deletes a specified file.

createFile(const std::string& fileName): Creates a new empty file with the specified name.

searchFiles(const std::string& directory, const std::string& pattern): Recursively searches for files that match a pattern.

setPermissions(const std::string& path, const std::string& mode): Changes the permissions of a file or directory.

parseCommand(const std::string& command): Splits a command string into tokens for easier processing.

**Main Function:**

Command Loop: The user is continuously prompted to input commands, which the program interprets and executes.

**List Files:**

Command: list

Lists all files and directories in the current directory.

**Change Directory:**

Command: cd folder1

Changes the current working directory to "folder1".

**Copy File:**

Command: copy file1.txt file2.txt

Copies "file1.txt" to "file2.txt".

**move (Rename) File:**

Command: move file1.txt newname.txt

Renames "file1.txt" to "newname.txt".

**Delete File:**

Command: delete file1.txt

Deletes "file1.txt".

**Create File:**

Command: create newfile.txt

Creates an empty file named "newfile.txt".

**Search Files:**

Command: search file

Searches for files that contain "file" in their name in the current directory and subdirectories.

**Set File Permissions:**

Command: chmod file1.txt 755

Changes the permissions of "file1.txt" to 755.

**Exit:**

Command: exit

Exits the program.

Explanation of Key Functions:

listFiles():

Uses the opendir and readdir functions to list all files and directories in the current working directory.

changeDirectory():

Uses chdir to change the current working directory and updates currentDirectory accordingly.

**copyFile():**

Uses file streams to copy the contents of one file to another.

**moveFile():**

Uses the rename function to move or rename a file.

**deleteFile():**

Uses the unlink function to delete a file.

**createFile():**

Creates an empty file using an ofstream.

**searchFiles():**

Recursively searches through directories using readdir and checks if file names contain the specified pattern.

**setPermissions():**

Uses the chmod function to change file permissions based on a specified mode.

**parseCommand():**

Uses istringstream to split the input command into individual tokens (words) for easier parsing.