FILE SYSTEM SIMULATOR

QUESTION:-

Data Structures: Tree (linked list representation) for directory structure, linked list for file information within directories.

Functionality:

Create directories and files.

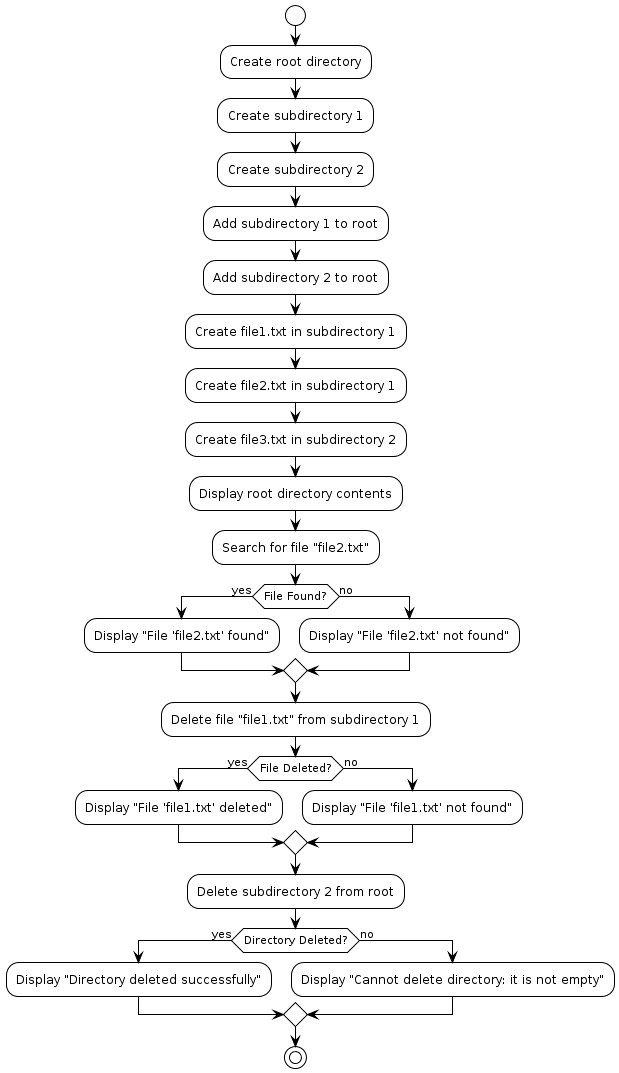
View directory contents (list files and subdirectories).

Navigate into subdirectories.

Delete files and directories (handle potential issues like non-empty directories).

Search for files by name.

FLOWCHART:-



MODULAR CODE

File.h:-

#ifndef LLTREE\_H

#define LLTREE\_H

// Structure for a file

struct File {

char name[100];

struct File\* next;

};

typedef struct File File;

// Structure for a directory

struct Directory {

char name[100];

struct Directory\* subdirectories;

struct File\* files;

struct Directory\* next;

};

typedef struct Directory Directory;

// Function declarations

Directory\* createDirectory(char\* name);

File\* createFile(char\* name);

void addDirectory(Directory\*\* root, char\* name);

void addFile(Directory\* directory, char\* name);

void viewDirectoryContents(Directory\* directory);

Directory\* navigateToSubdirectory(Directory\* root, char\* name);

void deleteFile(Directory\* directory, char\* name);

void deleteDirectory(Directory\*\* root, char\* name);

File\* searchFile(Directory\* directory, char\* name);

#endif /\* LLTREE\_H \*/

file.c:-

#include "lltree.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

Directory\* createDirectory(char\* name) {

Directory\* newDirectory = (Directory\*)malloc(sizeof(Directory));

if (newDirectory == NULL) {

printf("Memory allocation failed.\n");

exit(EXIT\_FAILURE);

}

strcpy(newDirectory->name, name);

newDirectory->subdirectories = NULL;

newDirectory->files = NULL;

newDirectory->next = NULL;

return newDirectory;

}

File\* createFile(char\* name) {

File\* newFile = (File\*)malloc(sizeof(File));

if (newFile == NULL) {

printf("Memory allocation failed.\n");

exit(EXIT\_FAILURE);

}

strcpy(newFile->name, name);

newFile->next = NULL;

return newFile;

}

void addDirectory(Directory\*\* root, char\* name) {

Directory\* newDirectory = createDirectory(name);

newDirectory->next = \*root;

\*root = newDirectory;

}

void addFile(Directory\* directory, char\* name) {

File\* newFile = createFile(name);

newFile->next = directory->files;

directory->files = newFile;

}

void viewDirectoryContents(Directory\* directory) {

printf("Directory: %s\n", directory->name);

printf("Files:\n");

File\* filePtr = directory->files;

while (filePtr != NULL) {

printf("- %s\n", filePtr->name);

filePtr = filePtr->next;

}

printf("Subdirectories:\n");

Directory\* subdirectoryPtr = directory->subdirectories;

while (subdirectoryPtr != NULL) {

printf("- %s\n", subdirectoryPtr->name);

subdirectoryPtr = subdirectoryPtr->next;

}

}

Directory\* navigateToSubdirectory(Directory\* root, char\* name) {

Directory\* current = root->subdirectories;

while (current != NULL) {

if (strcmp(current->name, name) == 0)

return current;

current = current->next;

}

printf("Subdirectory '%s' not found.\n", name);

return NULL;

}

void deleteFile(Directory\* directory, char\* name) {

File\* current = directory->files;

File\* prev = NULL;

while (current != NULL && strcmp(current->name, name) != 0) {

prev = current;

current = current->next;

}

if (current == NULL) {

printf("File '%s' not found.\n", name);

return;

}

if (prev == NULL)

directory->files = current->next;

else

prev->next = current->next;

free(current);

}

void deleteDirectory(Directory\*\* root, char\* name) {

Directory\* current = \*root;

Directory\* prev = NULL;

while (current != NULL && strcmp(current->name, name) != 0) {

prev = current;

current = current->next;

}

if (current == NULL) {

printf("Directory '%s' not found.\n", name);

return;

}

if (prev == NULL)

\*root = current->next;

else

prev->next = current->next;

free(current);

}

File\* searchFile(Directory\* directory, char\* name) {

File\* current = directory->files;

while (current != NULL) {

if (strcmp(current->name, name) == 0)

return current;

current = current->next;

}

return NULL;

}

mainfile.c:-

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include "lltree.h"

int main() {

clock\_t start, end;

double cpu\_time\_used;

Directory\* root = createDirectory("root");

// Creating directories and files

addDirectory(&root, "folder1");

addFile(root, "file1.txt");

// Viewing directory contents

viewDirectoryContents(root);

// Navigating to subdirectory

Directory\* subdirectory = navigateToSubdirectory(root, "folder1");

if (subdirectory != NULL)

printf("Navigated to subdirectory: %s\n", subdirectory->name);

// Deleting file

deleteFile(root, "file1.txt");

// Deleting directory

deleteDirectory(&root, "folder1");

// Measure time taken

start = clock();

// Perform operations here...

end = clock();

cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;

printf("Time taken: %f seconds\n", cpu\_time\_used);

return 0;

}

EXPLINATION

Directory Structure:

The directory structure can be represented using a tree data structure, where each node represents a directory and its children represent subdirectories and files.

Each node in the tree will contain information such as the directory or file name, and a reference to its parent directory.

You can implement the tree using a linked list representation, where each node has a pointer to its parent directory and a list of child directories/files.

Creating Directories and Files:

To create a directory, you would add a new node to the tree structure under the desired parent directory.

To create a file, you would add information about the file to the list of files within a directory.

Viewing Directory Contents:

To view the contents of a directory, you would traverse the tree starting from the specified directory node and list the files and subdirectories it contains.

Navigating into Subdirectories:

Navigating into a subdirectory involves moving to the corresponding node in the tree structure representing that subdirectory.

Deleting Files and Directories:

Deleting a file involves removing its information from the list of files within its parent directory.

Deleting a directory requires recursively deleting all its contents (files and subdirectories), ensuring to handle non-empty directories properly.

Searching for Files by Name:

Searching for a file involves traversing the tree and looking for files with matching names within the specified directory and its subdirectories.

By implementing these functionalities, you can create a file system simulator that mimics the basic operations of a real file system. Additionally, you can extend the simulator with features like file permissions, file metadata, file copying/moving, and error handling to make it more robust and realistic.

OUTPUT:-

