BHAVANA

03/05/2024

PROJECT 3:

File System Simulator:

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.

IMPLEMENTATION:

File_system.c

#include "file_system.h" // Including header file for file system functionality

#include <stdio.h> // Including standard input-output library

#include <stdlib.h> // Including standard library for memory allocation and other functions

#include <string.h> // Including string manipulation functions

// Global variables for root directory and current directory

Directory* rootDirectory = NULL; // Declaration and initialization of root directory pointer

Directory* currentDirectory = NULL; // Declaration and initialization of current directory pointer

```
void initializeFileSystem() {
    rootDirectory = (Directory*)malloc(sizeof(Directory)); // Allocating memory for root directory
    rootDirectory->name = strdup("root"); // Setting name of root directory
    rootDirectory->parent = NULL; // Setting parent directory of root directory to NULL
    rootDirectory->subdirectories = NULL; // Setting subdirectories of root directory to NULL
    rootDirectory->files = NULL; // Setting files in root directory to NULL
    rootDirectory->next = NULL; // Setting next directory pointer to NULL
    currentDirectory = rootDirectory; // Setting current directory to root directory
}

void createDirectory(const char* name) {
    Directory* newDirectory = (Directory*)malloc(sizeof(Directory)); // Allocating memory for new directory
    newDirectory->name = strdup(name); // Setting name of new directory
    newDirectory->parent = currentDirectory; // Setting parent directory of new directory
    newDirectory->subdirectories = NULL; // Setting subdirectories of new directory to NULL
    newDirectory->files = NULL; // Setting files in new directory to NULL
    newDirectory->next = NULL; // Setting next directory pointer to NULL

    if (currentDirectory->subdirectories == NULL) { // Checking if current directory has no subdirectories
        currentDirectory->subdirectories = newDirectory; // Setting new directory as the first subdirectory
    } else { // If current directory has existing subdirectories
```

```
        Directory* temp = currentDirectory->subdirectories; // Temporary pointer to traverse existing subdirectories
        while (temp->next != NULL) { // Looping until last subdirectory is reached
            temp = temp->next; // Moving to next subdirectory
        }
        temp->next = newDirectory; // Adding new directory to the end of subdirectories list
    }
}

void createFile(const char* name, int size) {
    File* newFile = (File*)malloc(sizeof(File)); // Allocating memory for new file
    newFile->name = strdup(name); // Setting name of new file
    newFile->size = size; // Setting size of new file
    newFile->next = NULL; // Setting next file pointer to NULL

    if (currentDirectory->files == NULL) { // Checking if current directory has no files
        currentDirectory->files = newFile; // Setting new file as the first file in directory
    } else { // If current directory has existing files
        File* temp = currentDirectory->files; // Temporary pointer to traverse existing files
        while (temp->next != NULL) { // Looping until last file is reached
            temp = temp->next; // Moving to next file
        }
        temp->next = newFile; // Adding new file to the end of files list
    }
```

```
}

void viewDirectoryContents(const char* directoryName) {
    Directory* temp = currentDirectory->subdirectories; // Temporary pointer to traverse subdirectories
    while (temp != NULL) { // Looping through subdirectories
        printf("Directory: %s\n", temp->name); // Printing name of each subdirectory
        temp = temp->next; // Moving to next subdirectory
    }

    File* fileTemp = currentDirectory->files; // Temporary pointer to traverse files
    while (fileTemp != NULL) { // Looping through files
        printf("File: %s, Size: %d\n", fileTemp->name, fileTemp->size); // Printing name and size of each file
        fileTemp = fileTemp->next; // Moving to next file
    }
}

void navigateIntoDirectory(const char* directoryName) {
    Directory* temp = currentDirectory->subdirectories; // Temporary pointer to traverse subdirectories
    while (temp != NULL) { // Looping through subdirectories
        if (strcmp(temp->name, directoryName) == 0) { // Checking if directory name matches
            currentDirectory = temp; // Changing current directory to matched directory
            return; // Exiting function
        }
```

```
        temp = temp->next; // Moving to next subdirectory
    }
    printf("Directory not found.\n"); // Printing message if directory not found
}

void deleteFile(const char* fileName) {
    File* temp = currentDirectory->files; // Temporary pointer to traverse files
    File* prev = NULL; // Pointer to track previous file in list
    while (temp != NULL) { // Looping through files
        if (strcmp(temp->name, fileName) == 0) { // Checking if file name matches
            if (prev == NULL) { // If file to be deleted is the first file in list
                currentDirectory->files = temp->next; // Update files list to skip first file
            } else { // If file to be deleted is not the first file in list
                prev->next = temp->next; // Adjusting previous file's next pointer to skip deleted file
            }
            free(temp->name); // Freeing memory allocated for file name
            free(temp); // Freeing memory allocated for file structure
            return; // Exiting function
        }
        prev = temp; // Moving previous pointer to current file
        temp = temp->next; // Moving to next file
    }
    printf("File not found.\n"); // Printing message if file not found
}
```

```
void deleteDirectory(const char* directoryName) {
    Directory* temp = currentDirectory->subdirectories; // Temporary pointer to traverse subdirectories
    Directory* prev = NULL; // Pointer to track previous directory in list
    while (temp != NULL) { // Looping through subdirectories
        if (strcmp(temp->name, directoryName) == 0) { // Checking if directory name matches
            if (prev == NULL) { // If directory to be deleted is the first directory in list
                currentDirectory->subdirectories = temp->next; // Update subdirectories list to skip first directory
            } else { // If directory to be deleted is not the first directory in list
                prev->next = temp->next; // Adjusting previous directory's next pointer to skip deleted directory
            }
            free(temp->name); // Freeing memory allocated for directory name
            free(temp); // Freeing memory allocated for directory structure
            return; // Exiting function
        }
        prev = temp; // Moving previous pointer to current directory
        temp = temp->next; // Moving to next directory
    }
    printf("Directory not found.\n"); // Printing message if directory not found
}

void searchFile(const char* fileName) {
    File* temp = currentDirectory->files; // Temporary pointer to traverse files
    while (temp != NULL) { // Looping through files
```

```
        if (strcmp(temp->name, fileName) == 0) { // Checking if file name
matches
            printf("File found: %s, Size: %d\n", temp->name, temp->size); //
Printing name and size of file
            return; // Exiting function
        }
        temp = temp->next; // Moving to next file
    }
    printf("File not found.\n"); // Printing message if file not found
}
```

EXPLANATION:

This code appears to be a simple implementation of a file system in C. Let's break down what each function does:

1. **initializeFileSystem()**: Initializes the file system by creating the root directory.

2. **createDirectory(const char* name)**: Creates a new directory with the specified name under the current directory.

3. **createFile(const char* name, int size)**: Creates a new file with the specified name and size under the current directory.

4. **viewDirectoryContents(const char* directoryName)**: Displays the contents of the current directory, including subdirectories and files.

5. **navigateIntoDirectory(const char* directoryName)**: Changes the current directory to the one with the specified name.

6. **deleteFile(const char* fileName)**: Deletes the file with the specified name from the current directory.

7. **deleteDirectory(const char* directoryName)**: Deletes the directory with the specified name from the current directory.

8. **searchFile(const char* fileName)**: Searches for a file with the specified name in the current directory.

Overall, this code provides basic functionality for managing directories and files within a file system. It uses linked lists to store subdirectories and files within each directory. However, there are a few potential improvements and considerations:

- Error handling: The code assumes that memory allocation always succeeds and that directory or file names are always provided. It's good practice to add error handling for these cases.

- Memory management: The code allocates memory for directory and file names using `malloc()` and `strdup()`, but it doesn't free this memory when directories or files are deleted. Proper memory management is important to avoid memory leaks.

- Directory traversal: The code traverses directories and files using linear search, which may not be efficient for large file systems. Consider using more efficient data structures or algorithms for directory traversal.

- Command-line interface: It could be useful to integrate this file system implementation with a command-line interface to interactively perform file system operations.

File_system.h:

```
#ifndef FILE_SYSTEM_H

#define FILE_SYSTEM_H
```

```
#include <stdbool.h> // Including header file for boolean data type

// Structures
typedef struct File { // Defining structure for a file
    char* name;         // Name of the file
    int size;           // Size of the file
    struct File* next;  // Pointer to the next file in the list
} File;

typedef struct Directory { // Defining structure for a directory
    char* name;                // Name of the directory
    struct Directory* parent;   // Pointer to the parent directory
    struct Directory* subdirectories;  // Pointer to the first subdirectory
    File* files;               // Pointer to the first file in the directory
    struct Directory* next;    // Pointer to the next directory in the list
} Directory;

// Function declarations
void initializeFileSystem();  // Function prototype to initialize the file system
void createDirectory(const char* name);  // Function prototype to create a new directory
void createFile(const char* name, int size);  // Function prototype to create a new file
void viewDirectoryContents(const char* directoryName);  // Function prototype to view contents of a directory
void navigateIntoDirectory(const char* directoryName);  // Function prototype to navigate into a directory
void deleteFile(const char* fileName);  // Function prototype to delete a file
```

void deleteDirectory(const char* directoryName); // Function prototype to delete a directory

void searchFile(const char* fileName); // Function prototype to search for a file

#endif // End of preprocessor directive

EXPLANATION:

The provided header file `file_system.h` defines the structures and function prototypes for a basic file system implementation. Let's break down its components:

1. **Structures**:

   - `File`: Represents a file in the file system. It contains fields for the file name, size, and a pointer to the next file in the list.

   - `Directory`: Represents a directory in the file system. It contains fields for the directory name, a pointer to the parent directory, pointers to subdirectories and files within the directory, and a pointer to the next directory in the list.

2. **Function Declarations**:

   - `initializeFileSystem()`: Initializes the file system.

   - `createDirectory(const char* name)`: Creates a new directory.

   - `createFile(const char* name, int size)`: Creates a new file.

   - `viewDirectoryContents(const char* directoryName)`: Views the contents of a directory.

   - `navigateIntoDirectory(const char* directoryName)`: Navigates into a directory.

   - `deleteFile(const char* fileName)`: Deletes a file.

   - `deleteDirectory(const char* directoryName)`: Deletes a directory.

   - `searchFile(const char* fileName)`: Searches for a file.

3. **Preprocessor Directive**:

- `#ifndef FILE_SYSTEM_H` and `#define FILE_SYSTEM_H`: These lines ensure that the content within the header file is only included once in a compilation unit, preventing duplicate definitions.

- `#endif`: Marks the end of the preprocessor directive block.

Overall, this header file provides the necessary definitions and declarations for implementing and interacting with a file system. It's a good practice to encapsulate such declarations in header files for modularization and reusability.

Main.c:

```
#include <stdio.h>
#include <time.h> // Include the time.h header for clock function
#include "file_system.h"

// Declare currentDirectory as an extern variable
extern Directory* currentDirectory;

int main() {
    clock_t start, end; // Variables to store start and end CPU time

    // Initialize the file system
    initializeFileSystem();

    // Create some directories and files
    createDirectory("documents");
    createDirectory("images");
    createFile("document1.txt", 100);
    createFile("image1.jpg", 500);
```

```
// Print the current directory contents
printf("Current directory contents:\n");
viewDirectoryContents(currentDirectory->name);

// Start measuring CPU time
start = clock();

// Navigate into 'documents' directory and create a file
printf("\nNavigating into 'documents' directory...\n");
navigateIntoDirectory("documents");
createFile("document2.txt", 200);

// Stop measuring CPU time
end = clock();

// Calculate CPU time elapsed
double cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

// Print the elapsed CPU time
printf("\nCPU time used: %.6f seconds\n", cpu_time_used);

// Print the current directory contents again
printf("\nCurrent directory contents:\n");
viewDirectoryContents(currentDirectory->name);

// Search for a file
```

```
    printf("\nSearching for 'image1.jpg'...\n");
    searchFile("image1.jpg");

    // Delete a file
    printf("\nDeleting 'document1.txt'...\n");
    deleteFile("document1.txt");

    // Print the current directory contents one more time
    printf("\nCurrent directory contents:\n");
    viewDirectoryContents(currentDirectory->name);

    return 0;
}
```

EXPLANATION:

The `main()` function provided demonstrates the usage of the file system functions defined in `file_system.h`. Here's a breakdown of what it does:

1. **Include Statements**: Standard headers like `stdio.h` and `time.h` are included. The `file_system.h` header, which contains the declarations for file system functions and structures, is also included.

2. **External Variable Declaration**: `extern Directory* currentDirectory;` declares `currentDirectory` as an external variable, meaning its definition is provided elsewhere (in `file_system.c`, presumably).

3. **Main Function**:

   - **File System Initialization**: The file system is initialized using `initializeFileSystem()`.

- **Directory and File Creation**: Directories ("documents" and "images") and files ("document1.txt" and "image1.jpg") are created using `createDirectory()` and `createFile()` functions.

- **Viewing Current Directory Contents**: The contents of the current directory are printed using `viewDirectoryContents()` function.

- **Navigation and File Creation**: The program navigates into the "documents" directory, creates a file ("document2.txt"), and measures CPU time using `clock()` before and after the operation.

- **Printing CPU Time**: The elapsed CPU time for the previous operation is printed.

- **Viewing Current Directory Contents Again**: The contents of the current directory are printed again.

- **File Searching**: The program searches for a file ("image1.jpg") using `searchFile()` function.

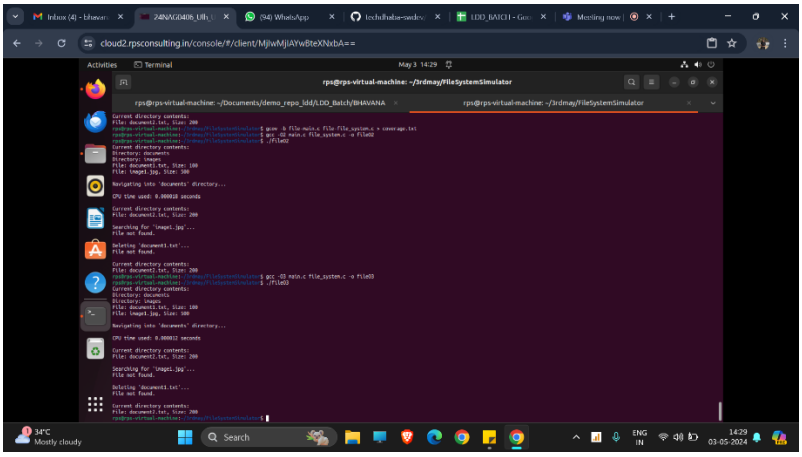
- **File Deletion**: A file ("document1.txt") is deleted using `deleteFile()` function.

- **Viewing Current Directory Contents Once More**: The contents of the current directory are printed again after the deletion.

Overall, this `main()` function demonstrates various file system operations like directory and file creation, navigation, searching, and deletion, along with measuring CPU time for an operation.

OUTPUT:

OPTIMIZING THE CODE USING COMMANDS:



CREATING ANALYSIS AND COVERAGE REPORT: