Implementing a File System in C

You are required to implement a basic file system simulator in C that can manage directories and files.
The file system should have the following features:
1. A directory can contain other subdirectories and files.
2. Files and directories can be added, but directories cannot contain two subdirectories or files with the same name.
3. The file system should start with a root directory.

The file system must support the following operations:
1. Change directory: (a) Move the head pointer to a child directory, given the child directory name. (b) Move to the parent directory.
2. Make Directory: Create a subdirectory with given name under the current directory.
3. Create file: Create a file with given name in current directory.
4. List contents: Print the contents of the current directory.

Input Format

None

Constraints

None

Output Format

None

C

```c
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// ------------------------ Structure Definitions ------------------------

typedef struct File {
  char *filename;
  struct File *next;
} File;

typedef struct Directory {
  char *directoryName;
  struct Directory *subdirectories; // Head of subdirectories linked list
```

```c
  struct Directory *next;           // Next sibling directory
  File *files;                      // Head of files linked list
  struct Directory *parent;         // Pointer to parent directory
} Directory;

typedef struct FileSystem {
  Directory *rootDirectory;
} FileSystem;

// ------------------------ Function Prototypes ------------------------

Directory *create_directory(const char *directoryName, Directory *parent);
Directory *get_subdirectory(Directory *parent, const char *dirName);
Directory *add_subdirectory(Directory *dir, const char *directoryName);
int add_file(Directory *dir, const char *filename);
void print_directory_contents(Directory *dir);
void print_directory_hierarchy(Directory *dir, int depth);
void print_filesystem(FileSystem *fs);
void cleanup_filesystem(FileSystem *fs);

FileSystem *create_filesystem(const char *rootName);
void trim_newline(char *str);
void cleanup_directory(Directory *dir);
void cleanup_file(File *file);

// ---------------------- Function Implementations ------------------------

Directory *create_directory(const char *directoryName, Directory *parent) {
  Directory *newDir = (Directory *)malloc(sizeof(Directory));
  if (!newDir) {
    perror("Failed to allocate memory for directory");
    exit(EXIT_FAILURE);
  }
  newDir->directoryName = strdup(directoryName);
  newDir->subdirectories = NULL;
  newDir->files = NULL;
  newDir->parent = parent;
  newDir->next = NULL;
  return newDir;
}

Directory *get_subdirectory(Directory *parent, const char *dirName) {
  Directory *subDir = parent->subdirectories;
  while (subDir) {
    if (strcmp(subDir->directoryName, dirName) == 0) {
      return subDir;
    }
    subDir = subDir->next;
  }
  return NULL;
}

Directory *add_subdirectory(Directory *dir, const char *directoryName) {
  if (get_subdirectory(dir, directoryName)) {
    return NULL; // Directory already exists
  }
  Directory *newDir = create_directory(directoryName, dir);
  newDir->next = dir->subdirectories;
  dir->subdirectories = newDir;
  return newDir;
}

int add_file(Directory *dir, const char *filename) {
  File *current = dir->files;
  while (current) {
    if (strcmp(current->filename, filename) == 0) {
      return 0; // File already exists
    }
    current = current->next;
  }
  File *newFile = (File *)malloc(sizeof(File));
  newFile->filename = strdup(filename);
  newFile->next = dir->files;
```

```c
    dir->files = newFile;
    return 1;
}

void print_directory_contents(Directory *dir) {
    printf("Directory: %s\n", dir->directoryName);

    // List subdirectories
    Directory *currentDir = dir->subdirectories;
    if (currentDir) {
        printf("Subdirectories:\n");
        while (currentDir) {
            printf("- %s\n", currentDir->directoryName);
            currentDir = currentDir->next;
        }
    } else {
        printf("No subdirectories\n");
    }

    // List files
    File *currentFile = dir->files;
    if (currentFile) {
        printf("Files:\n");
        while (currentFile) {
            printf("- %s\n", currentFile->filename);
            currentFile = currentFile->next;
        }
    } else {
        printf("No files\n");
    }
}

void print_directory_hierarchy(Directory *dir, int depth) {
    for (int i = 0; i < depth; i++) {
        printf("    ");
    }
    printf("%s/\n", dir->directoryName);

    Directory *currentDir = dir->subdirectories;
    while (currentDir) {
        print_directory_hierarchy(currentDir, depth + 1);
        currentDir = currentDir->next;
    }

    File *currentFile = dir->files;
    while (currentFile) {
        for (int i = 0; i < depth + 1; i++) {
            printf("    ");
        }
        printf("%s\n", currentFile->filename);
        currentFile = currentFile->next;
    }
}

void print_filesystem(FileSystem *fs) {
    printf("File System Contents:\n");
    print_directory_hierarchy(fs->rootDirectory, 0);
}

void cleanup_file(File *file) {
    while (file) {
        File *nextFile = file->next;
        free(file->filename);
        free(file);
        file = nextFile;
    }
}

void cleanup_directory(Directory *dir) {
    while (dir) {
        Directory *nextDir = dir->next;
        cleanup_directory(dir->subdirectories);
        cleanup_file(dir->files);
```

```c
    free(dir->directoryName);
    free(dir);
    dir = nextDir;
  }
}

void cleanup_filesystem(FileSystem *fs) {
  if (fs) {
    cleanup_directory(fs->rootDirectory);
    free(fs);
  }
}

FileSystem *create_filesystem(const char *rootName) {
  FileSystem *fs = (FileSystem *)malloc(sizeof(FileSystem));
  if (!fs) {
    perror("Failed to allocate memory for file system");
    exit(EXIT_FAILURE);
  }
  fs->rootDirectory = create_directory(rootName, NULL);
  return fs;
}

void trim_newline(char *str) {
  if (str) {
    size_t len = strlen(str);
    if (len > 0 && str[len - 1] == '\n') {
      str[len - 1] = '\0';
    }
  }
}

// ---------------------------- Main Function ----------------------------

int main() {
  FileSystem *fs = create_filesystem("root");
  Directory *currentDir = fs->rootDirectory;
  char input[256];

  printf("Welcome to the In-Memory File System!\n");
  printf("Available commands: cd <dir>, mkdir <dir>, touch <file>, ls, print, "
         "exit\n");

  while (1) {
    printf(">> ");
    if (!fgets(input, sizeof(input), stdin)) {
      printf("Error reading input. Exiting.\n");
      break;
    }
    trim_newline(input);

    char *command = strtok(input, " ");
    char *argument = strtok(NULL, " ");

    if (strcmp(command, "cd") == 0) {
      if (argument && strcmp(argument, "..") == 0) {
        if (currentDir->parent)
          currentDir = currentDir->parent;
      } else if (argument) {
        Directory *nextDir = get_subdirectory(currentDir, argument);
        currentDir = nextDir ? nextDir : currentDir;
      }
    } else if (strcmp(command, "mkdir") == 0 && argument) {
      add_subdirectory(currentDir, argument);
    } else if (strcmp(command, "touch") == 0 && argument) {
      add_file(currentDir, argument);
    } else if (strcmp(command, "ls") == 0) {
      print_directory_contents(currentDir);
    } else if (strcmp(command, "print") == 0) {
      print_filesystem(fs);
    } else if (strcmp(command, "exit") == 0) {
      break;
    } else {
```

```
235         printf("Unknown command\n");
236       }
237     }
238
239     cleanup_filesystem(fs);
240     printf("Goodbye!\n");
241     return 0;
242 }
```

Line: 242 Col: 2

Upload Code as File Test against custom input Run Code Submit Code

No sample test-cases for this question. Please test your code against custom input.