OS Task

Code :

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <dirent.h>
#include <errno.h>
```

Explanation :

```
#include <stdio.h>:
 Includes standard input-output functions.
#include <stdlib.h>:
 Includes standard library functions like memory allocation and
process control.
#include <unistd.h>:
 Includes POSIX operating system API for system calls like chmod,
mkdir, etc.
#include <sys/stat.h>:
 Includes definitions for file status (e.g., file permissions).
#include <sys/types.h>:
Includes definitions for various data types used in system calls.
#include <dirent.h>:
Includes directory entry structures and functions for directory
manipulation.
#include <errno.h>:
 Includes definitions for error numbers.
```

listFiles

```c
void listFiles(char *path)
{
    DIR *dir;
    struct dirent *entry;

    if ((dir = opendir(path)) != NULL)
    {
        printf("Files in directory %s:\n", path);
        while ((entry = readdir(dir)) != NULL)
        {
            printf("%s\n", entry->d_name);
        }
        closedir(dir);
    }
    else
    {
        perror("Unable to open directory");
    }
}
```

Explanation :

```
// Function Signature:
void listFiles(char *path): This function takes a char pointer path as
input. This path represents the directory whose files are to be
listed.

// Function Logic:

1-Opening the Directory:

DIR *dir;: Declares a pointer dir of type DIR (directory stream).
*** DIR is a  is a data type used in the Cto represent a directory
stream. typically used in conjunction with functions For directory
```

manipulation to facilitate reading directory contents and navigating the file system.
struct dirent *entry;: Declares a pointer entry of type struct dirent (directory entry).
dir = opendir(path);: Attempts to open the directory specified by path. The opendir function returns a pointer to the directory stream (DIR) if successful, or NULL if it fails to open the directory.

2-Processing Directory Contents:

if (dir != NULL): Checks if the directory was successfully opened.
printf("Files in directory %s:\n", path);: Prints a message indicating the directory being listed.
while ((entry = readdir(dir)) != NULL): Loops through each entry in the directory using the readdir function. It reads the next directory entry from the directory stream pointed to by dir.
printf("%s\n", entry->d_name);: Prints the name of each entry (file or directory) in the directory.

3-Closing the Directory:

closedir(dir);: Closes the directory stream pointed to by dir once all entries have been processed.

5- Error Handling:

else: Executes if the directory could not be opened (dir is NULL).
perror("Unable to open directory");: Prints an error message indicating the failure to open the directory, along with a system error message obtained from perror.

*** Overall Logic:
The function attempts to open the specified directory.
If successful, it iterates through each entry in the directory and prints its name.
After processing all entries, it closes the directory.
If the directory cannot be opened, it prints an error message.

changePermissions

```c
void changePermissions(char *file, mode_t mode)
{
    if (chmod(file, mode) == -1)
    {
        perror("chmod failed");
    }
    else
    {
        printf("Permissions changed successfully.\n");
    }
}
```

Explanation :

```
//Function Signature:
void changePermissions(char *file, mode_t mode): This function takes
two parameters:
file: A pointer to a string representing the path of the file whose
permissions are to be changed.
mode: A value of type mode_t representing the new permissions to be
set for the file.
//Function Logic:

1-Changing File Permissions:

chmod(file, mode): Attempts to change the permissions of the file
specified by file to the value specified by mode. The chmod function
returns -1 if an error occurs, and 0 if the operation is successful.

//Error Handling:

if (chmod(file, mode) == -1): Checks if the chmod function returned an
error (-1).
perror("chmod failed");: Prints an error message using perror
indicating that the chmod operation failed. The error message is based
on the current value of errno.
else: Executes if the chmod operation is successful.
```

```
printf("Permissions changed successfully.\n");: Prints a success
message indicating that the permissions were changed successfully.

***Overall Logic:
The function attempts to change the permissions of the file specified
by file to the value specified by mode.
If successful, it prints a success message.
If an error occurs, it prints an error message indicating the failure
to change permissions.
Usage:
This function can be called with a file path and the desired
permissions as arguments to change the permissions of a file.

***Note:
The mode parameter represents the file permissions and is typically
specified using octal notation (e.g., 0644 for read/write permissions
for owner and read-only permissions for others).
Permissions may include read, write, and execute permissions for the
file owner, group, and others.
```

createFile

```c
void createFile(char *file)
{
    if (creat(file, 0666) == -1)
    {
        perror("Failed to create file");
    }
    else
    {
        printf("File created successfully.\n");
    }
}
```

Explanation :

```
//Function Signature:

void createFile(char *file): This function takes a pointer to a string
file representing the path of the file to be created.
```

```
//Function Logic:

Creating the File:
creat(file, 0666): Attempts to create a new file with the specified
path (file) and permissions (0666). The permissions 0666 allow read
and write access for the owner, group, and others.
The creat function returns -1 if an error occurs, and the file
descriptor of the newly created file if successful.

//Error Handling:
if (creat(file, 0666) == -1): Checks if the creat function returned an
error (-1).
perror("Failed to create file");: Prints an error message using perror
indicating that the file creation failed. The error message is based
on the current value of errno.
else: Executes if the file creation is successful.
printf("File created successfully.\n");: Prints a success message
indicating that the file was created successfully.

 ***Overall Logic:
createFile attempts to create a file with the specified path and
permissions. It prints a success message if the operation is
successful or an error message if it fails

***Note:
The 0666 permission in createFile allows read and write access for all
users. Depending on the desired permissions, this value can be
adjusted accordingly.
```

deleteFile

```
void deleteFile(char *file)
{
    if (remove(file) == -1)
    {
        perror("Failed to delete file");
    }
    else
    {
        printf("File deleted successfully.\n");
    }
}
```

Explanation :

```
//Function Signature:

void deleteFile(char *file): This function takes a pointer to a string
file representing the path of the file to be deleted.

//Function Logic:

Deleting the File:
remove(file): Attempts to delete the file specified by file.
The remove function returns -1 if an error occurs, and 0 if the
operation is successful.

//Error Handling:

if (remove(file) == -1): Checks if the remove function returned an
error (-1).
perror("Failed to delete file");: Prints an error message using perror
indicating that the file deletion failed. The error message is based
on the current value of errno.
else: Executes if the file deletion is successful.
printf("File deleted successfully.\n");: Prints a success message
indicating that the file was deleted successfully.

***Overall Logic:
deleteFile attempts to delete a file with the specified path. It
prints a success message if the operation is successful or an error
message if it fails.
```

createDirectory

```c
void createDirectory(char *path)
{
    if (mkdir(path, 0777) == -1)
    {
        perror("Failed to create directory");
    }
    else
    {
        printf("Directory created successfully.\n");
    }
}
```

Explanation :

```
//Function Signature:

void createDirectory(char *path): This function takes a pointer to a
string path representing the path of the directory to be created.

//Function Logic:

Creating the Directory:
mkdir(path, 0777): Attempts to create a new directory with the
specified path (path) and permissions (0777). The permissions 0777
allow read, write, and execute access for the owner, group, and
others.
The mkdir function returns -1 if an error occurs, and 0 if the
operation is successful.

//Error Handling:

if (mkdir(path, 0777) == -1): Checks if the mkdir function returned an
error (-1).
perror("Failed to create directory");: Prints an error message using
perror indicating that the directory creation failed. The error
message is based on the current value of errno.
else: Executes if the directory creation is successful.
```

```
printf("Directory created successfully.\n");: Prints a success message indicating that the directory was created successfully.

***Overall Logic:
createDirectory attempts to create a directory with the specified path and permissions. It prints a success message if the operation is successful or an error message if it fails.

***Note:
The 0777 permission in createDirectory allows read, write, and execute access for all users. Depending on the desired permissions, this value can be adjusted accordingly.
```

deleteDirectory

```c
void deleteDirectory(char *path)
{
    if (rmdir(path) == -1)
    {
        perror("Failed to delete directory");
    }
    else
    {
        printf("Directory deleted successfully.\n");
    }
}
```

Explanation :

```
//Function Signature:

void deleteDirectory(char *path): This function takes a pointer to a string path representing the path of the directory to be deleted.

//Function Logic:

Deleting the Directory:
rmdir(path): Attempts to delete the directory specified by path.
```

```
The rmdir function returns -1 if an error occurs, and 0 if the operation is
successful.

//Error Handling:

if (rmdir(path) == -1): Checks if the rmdir function returned an error (-1).
perror("Failed to delete directory");: Prints an error message using perror
indicating that the directory deletion failed. The error message is based on the
current value of errno.
else: Executes if the directory deletion is successful.
printf("Directory deleted successfully.\n");: Prints a success message indicating
that the directory was deleted successfully.

***Overall Logic:
deleteDirectory attempts to delete a directory with the specified path. It prints
a success message if the operation is successful or an error message if it fails.
```

createSymbolicLink

```c
void createSymbolicLink(char *target, char *link)
{
    if (symlink(target, link) == -1)
    {
        perror("Failed to create symbolic link");
    }
    else
    {
        printf("Symbolic link created successfully.\n");
    }
}
```

Explanation :

```
//Function Signature:
void createSymbolicLink(char *target, char *link): This function takes
two parameters:
target: A pointer to a string representing the path of the target file
or directory for the symbolic link.
link: A pointer to a string representing the path of the symbolic link
to be created.
```

```
//Function Logic:

Creating the Symbolic Link:
symlink(target, link): Attempts to create a symbolic link named link
that points to the file or directory specified by target.
The symlink function returns -1 if an error occurs, and 0 if the
operation is successful.

//Error Handling:

if (symlink(target, link) == -1): Checks if the symlink function
returned an error (-1).
perror("Failed to create symbolic link");: Prints an error message
using perror indicating that the symbolic link creation failed. The
error message is based on the current value of errno.
else: Executes if the symbolic link creation is successful.
printf("Symbolic link created successfully.\n");: Prints a success
message indicating that the symbolic link was created successfully.

***Overall Logic:
1-The function attempts to create a symbolic link named link that
points to the file or directory specified by target.
2-If successful, it prints a success message.
3-If an error occurs, it prints an error message indicating the
failure to create the symbolic link.

***Note:
1-Symbolic links are references to other files or directories,
allowing you to access them indirectly. They can be used to create
shortcuts or aliases to files or directories.
2-The function relies on the standard error handling mechanism
provided by perror to print descriptive error messages in case of
failure.
```

Menu

```c
void displayMenu()
{
    printf("\nMenu:\n");
    printf("1. List files/directories\n");
    printf("2. Change permissions of a file/directory\n");
    printf("3. Make a file\n");
    printf("4. Delete a file\n");
    printf("5. Make a directory\n");
    printf("6. Delete a directory\n");
    printf("7. Create symbolic link\n");
    printf("8. Exit\n");
    printf("Enter your choice: ");
}

void runFileManager()
{
    char path[100], file[100], target[100], link[100];
    mode_t mode;
    int choice;

    while (1)
    {
        displayMenu();
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            printf("Enter directory path: ");
            scanf("%s", path);
            listFiles(path);
            break;
        case 2:
            printf("Enter file/directory path: ");
            scanf("%s", file);
            printf("Enter permissions (in octal): ");
            scanf("%o", (unsigned int *)&mode);
            changePermissions(file, mode);
```

```
            break;
        case 3:
            printf("Enter file name: ");
            scanf("%s", file);
            createFile(file);
            break;
        case 4:
            printf("Enter file name: ");
            scanf("%s", file);
            deleteFile(file);
            break;
        case 5:
            printf("Enter directory path: ");
            scanf("%s", path);
            createDirectory(path);
            break;
        case 6:
            printf("Enter directory path: ");
            scanf("%s", path);
            deleteDirectory(path);
            break;
        case 7:
            printf("Enter target file/directory path: ");
            scanf("%s", target);
            printf("Enter symbolic link path: ");
            scanf("%s", link);
            createSymbolicLink(target, link);
            break;
        case 8:
            printf("Exiting...\n");
            exit(0);
        default:
            printf("Invalid choice.\n");
        }
    }
}



int main()
{
    runFileManager();
    return 0;
}
```