Rylan Casanova

04/16/23

CST-315

Project 4 – File System Manager

**Explanation of Design:**

In this project, we were assigned to design a file system with a hierarchical directory structure. This can be implemented by using a struct to represent files, and a class to represent directories. The implementation will also include a set of 13 commands which are listed in the assignment instructions: The design for these 13 listed commands can be implemented as follows:

1. Create and rename directories:
   * Use the mkdir system call to create a new directory.
   * Use the rename system call to rename an existing directory.
2. Delete directories, with special handling of non-empty directories:
   * Use the rmdir system call to delete an empty directory.
   * Use the unlink system call to delete a file.
   * Use the remove function from the <stdio.h> library to recursively delete a non-empty directory
3. Create, rename, edit, and delete files:
   * Use the open system call to create or open an existing file.
   * Use the write system call to write data to a file.
   * Use the read system call to read data from a file.
   * Use the close system call to close a file.
   * Use the rename system call to rename an existing file
   * Use the unlink system call to delete a file.
4. When a file is created, a special flag will indicate the amount of data (bytes) to (randomly) generate for it:
   * Use the rand function from the <cstdlib> library to generate random data.
   * Use the write system call to write the generated data to the file.
5. Move files across directories:
   * Use the rename system call to move a file from one directory to another.
6. Duplicate files:
   * Use the open system call to open the original file.
   * Use the read system call to read the contents of the original file.
   * Use the open system call to create a new file with a different name
   * Use the write system call to write the contents of the original file to the new file.
   * Use the close system call to close both files.
7. Duplicate directories:
   * Use the mkdir system call to create a new directory with a different name.
   * Use the opendir function from the dirent.h library to open the original directory.
   * Use the readdir function to read the contents of the original directory.
   * Recursively duplicate all files and subdirectories in the original directory into the new directory.
   * Use the closedir function to close both directories.
8. Search for a file in a directory tree:
   * Use the opendir function to open the starting directory.
   * Use the readdir function to read the contents of the directory.
   * For each file or subdirectory in the directory, recursively search for the file in it.
   * Use the closedir function to close the directory.
9. Display a directory tree given a starting node:
   * Use the opendir function to open the starting directory.
   * Use the readdir function to read the contents of the directory.
   * For each file or subdirectory in the directory, recursively display its name and contents.
   * Use the closedir function to close the directory.
10. Get basic information about a file:
    * Use the stat system call to get basic information about a file, such as its size, permissions, and timestamps.
11. Get detailed information about a file (e.g. using a special flag):
    * Use the stat system call to get detailed information about a file, such as its inode number, number of hard links, and device ID.
12. Get basic information about a directory:
    * Get the name of the directory from the user.
    * Use the stat() system call to obtain the basic information about the directory (e.g., size, number of subdirectories and files).
    * Display the information to the user.
13. Get detailed information about a directory (e.g., using a flag):
    * Get the name of the directory from the user.
    * Use the stat() system call to obtain the detailed information about the directory (e.g., owner, permissions, creation/modification/access time, etc.)
    * Display the information to the user based on the flag provided

**Concept and Analysis:**

1. **Create and rename directories:**

I implemented two functions – void create\_directory() and void rename\_directory(). These functions are used to create and rename directories. The create\_directory() function takes user input for a directory name and creates a directory with the specified name. The rename\_directory() function takes user input for the old and new directory names and renames the directory with the old name to the new name. Both functions output success or error messages.

1. **Delete directories, with special handling of non-empty directories:**

I implemented the function void delete\_directory() which deletes a directory and all its contents recursively. It prompts the user for the directory name, opens the directory, and iterates over its contents. If a subdirectory is encountered, it recursively calls itself on that subdirectory. If a file is encountered, it is deleted. Finally, the function attempts to delete the empty directory. If successful, it prints a success message; otherwise, it prints an error message.

1. **Create, rename, edit, and delete files:**

Four functions were implemented for this functionality:

* void create\_file(): prompts the user for a filename and number of bytes to generate, creates a binary file with randomly generated bytes.
* void rename\_file(): prompts the user for an old and new file name, renames the file.
* void edit\_file(): prompts the user for a file name and text to write to the file, saves the text to the file.
* void delete\_file(): prompts the user for a filename and deletes the file if it exists.

1. **When a file is created, a special flag will indicate the amount of data (bytes) to (randomly) generate for it:**

I implemented a function void generate\_random\_data() which generates random data and writes it to a file. It prompts the user to enter a filename and size in bytes, then creates the file and fills it with randomly generated bytes. It uses the time and srand functions to seed the random number generator. Finally, it outputs a message indicating success or failure.

1. **Move files across directories:**

I implemented the void move\_file() function which moves a file by prompting the user to enter the name of the file to be moved and the directory to move it to. It uses the rename() function to move the file to the new directory, and if the function returns an error, it outputs an error message. If the move is successful, the program outputs a success message.

1. **Duplicate files:**

I added the function void duplicate\_file() which prompts the user for the names of the source and destination files, and then uses std::ifstream and std::ofstream to open the source and destination files, respectively. It then copies the contents of the source file to the destination file using std::ofstream::operator<< and std::istream::rdbuf. Finally, it closes both files and prints a success message.

1. **Duplicate directories:**

To implement duplication of directories, the void duplicate\_directory() function takes two arguments: source\_dir, which is the path to the directory that is to be duplicated, and target\_dir, which is the path to the directory that will contain the duplicate. The function first opens both the source and target directories using the opendir function. It then reads each entry in the source directory using the readdir function and creates a corresponding entry in the target directory using the same filename.

1. **Search for a file in a directory tree:**

The void search\_file() function starts by prompting the user to enter the name of the file to search for. It then opens the current directory using opendir(). It then iterates through all the entries in the directory using readdir(). If the entry is a subdirectory, the function changes the current working directory to that directory using chdir(), and calls itself recursively. If the entry is a regular file with a name matching the search term, the function prints out the path to the file using getcwd() and std::cout. Finally, the function closes the directory using closedir().

1. **Display a directory tree given a starting node:**

The implementation for void display\_directory() uses a recursive approach to traverse the directory tree, starting from the specified directory and printing out the names of all the files and directories it encounters, indented according to their level in the tree. The display\_directory() function simply prompts the user for the starting directory and calls the recursive display\_directory() function with the appropriate parameters

1. **Get basic information about a file:**

The get\_basic\_file\_info() function prompts the user to enter the name of a file they want information about, then uses the stat function to retrieve the file's size, owner UID, and last access time. It then prints out this information to the console.

1. **Get detailed information about a file (e.g. using a special flag):**

The get\_detailed\_file\_info() function uses the stat() function to get file information and then prints it out in a readable format. It displays the file size, owner user ID, owner group ID, permissions, and timestamps for the last access, modification, and status change.

1. **Get basic information about a directory:**

The get\_basic\_directory\_info() function first prompts the user for a directory path and checks if it exists and is a directory using the exists() and is\_directory() functions from the std::filesystem library. If the directory is valid, it then prints the name of the directory using the filename() function of the path class, the number of files in the directory by counting the distance between two directory\_iterator objects, and the size of the directory using the file\_size() function of a directory\_entry object created with the directory path.

1. **Get detailed information about a directory (e.g., using a flag):**

The get\_detailed\_directory\_info() function uses the C++17 filesystem library to get detailed information about the current working directory. The function prints the name, size, last modified time, and type (file or directory) of each item in the directory. The output is formatted using the std::setw() function to align the columns.

**Test and Validate:**

* **Basic Shell Implementation (define functions, generate UI, translate user input):**

**Text

Description automatically generated with medium confidenceText

Description automatically generated**

**A picture containing graphical user interface

Description automatically generatedA picture containing graphical user interface

Description automatically generatedA picture containing table

Description automatically generated**

**Text

Description automatically generated**

1. **Create and rename directories:**

**Graphical user interface, text, application, email

Description automatically generated**

**Graphical user interface

Description automatically generated**

1. **Delete directories, with special handling of non-empty directories:**

**Text

Description automatically generated**

1. **Create, rename, edit, and delete files:**

**Graphical user interface, text, application

Description automatically generated**

**Graphical user interface, text

Description automatically generated with medium confidence**

**Graphical user interface, text, application, email

Description automatically generated**

**Graphical user interface, text, application

Description automatically generated**

1. **When a file is created, a special flag will indicate the amount of data (bytes) to (randomly) generate for it:**

**Graphical user interface, text, application, email

Description automatically generated**

1. **Move files across directories:**

**Graphical user interface, text, application

Description automatically generated**

1. **Duplicate files:**

**Graphical user interface, text, application

Description automatically generated**

1. **Duplicate directories:**

**Graphical user interface, text, application, email

Description automatically generated**

**Graphical user interface, text, application, email

Description automatically generated**

1. **Search for a file in a directory tree:**

**Text

Description automatically generated**

1. **Display a directory tree given a starting node:**

**Text

Description automatically generated**

1. **Get basic information about a file:**

**Graphical user interface, text

Description automatically generated**

1. **Get detailed information about a file (e.g. using a special flag):**

**Text

Description automatically generated**

1. **Get basic information about a directory:**

**Graphical user interface, text, application

Description automatically generated**

1. **Get detailed information about a directory (e.g., using a flag):**

**Text

Description automatically generated**