SystemHW1

Sezer Demir

March 2021

1 Requirements

For this homework I achieved all the requirements those are mentioned in homework pdf file. I didn't use any library functions except malloc, calloc, and free. There is no warning that comes from -Wall flag or memory leak showed by valgrind including CTRL-C interruption signal.

2 Design Decisions

a.) To print all the files those are found by the program I needed to store their associated full-path alongside their file names, depth, and types. Therefore I used a linked list to achieve that. Each node stores their names, types, paths, and depths. Depth indicates how many times program must have opened directory to reach the found file.

b.) I used another linked list structure to store all the pointers ,those are dynamically allocated, because in case of user interrupt the program by pressing CTRL-C.I should have to return all the resources back.To achieve that each node, that named as clean(another structure), stores those pointers, allocated throughout the program, in a void pointer. So I could release them if a interruption signal comes. When interruption signal comes I free all the clean nodes, so with that I release all the resources.

c.) I also checked arguments to make sure that they're entered in a proper form. If user types any invalid argument program gives a warning message and terminates itself. Some argument rules are following:

1) -b: takes just positive numbers (Example: 123123 valid, -123123 or 959awf569 invalid).
2) -l: takes just positive numbers (Example: 123123 valid, -123123 or 959awf569 invalid).

3) -f: file names cannot include '/' character in UNIX, except that character you can use any character.

4) -t: can take only one of the 'd', 's', 'b', 'c', 'f', 'p', 'l' characters.

5) -w: you can enter any character for that one, if there is not such a directory it prints a warning message and terminates the program.

3 Algorithm

Finding Files

1. Open next file.
2. Check all the file conditions those are provided by user with flags.
3. If current file satisfies all the conditions create a new node and fill all necessary information in it like file name, path, type, and depth.
4. Repeat that process until traverse all the files under the directory that is provided by user.

Printing Founded Files

To print all the files in a proper form it checkes nodes, those have information about found files, one by one. Of course, it starts checking directories from the base directory, that provided by user, to found file's directory. To achieve that it parses directory names of found file's path. And it prints sub directory name and whenever it prints a directory name, I just store it's name in a linked list and whenever I need to print another directory, first I check that the directory printed before or not by checking that linked list, that stores all the printed directory names. Therefore it could avoid print a directory name more than once. Whenever a directory name is printed, I checked that there is any found file at same directory that I parsed. If there is it prints it too. At the end it prints the found file name and repeat all those steps until all found files in linked list are printed. The sign before the names of the files is created according to their depth information.

Lets say user wanted to find a file under "/home/user/Downloads" directory. And one of the files is under "/home/user/Downloads/sss/ddd/" path. At first iteration it parses the "sss" directory name, print it, and checks all the other found files those have "/home/user/Downloads/sss/" path and prints them too. Then adds "sss" name and it's depth as 1 (since it is right under Downloads directory. Depth is 2 for ddd for example.) to a linked list to mark it as printed directory. At the same time it marks all the found files by assigning their "found" variable to 1, so it can check whenever a directory or file will be printed before or not. After that, it parses "ddd" and so on. And repeats those processes for all found files.

4 Input and Output Examples

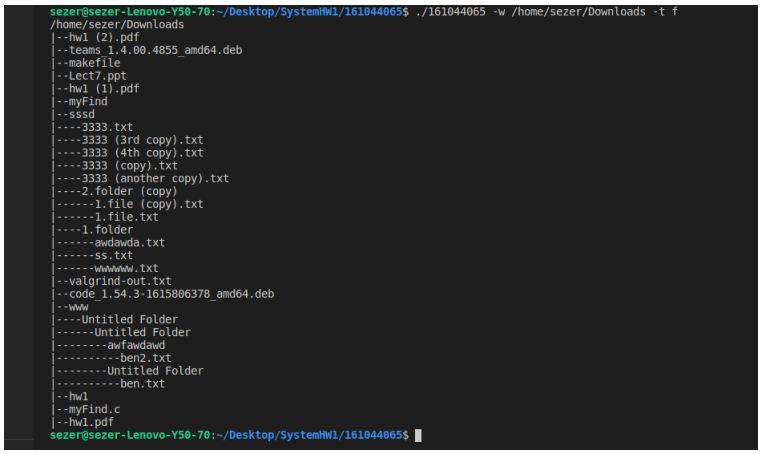

Figure 1: Valid Example 1

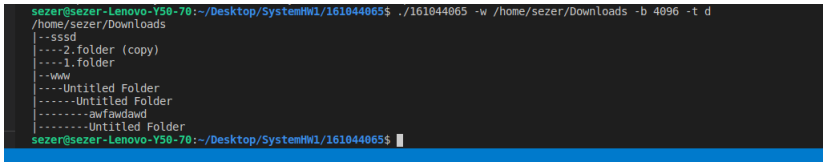

Figure 2: Valid Example 2

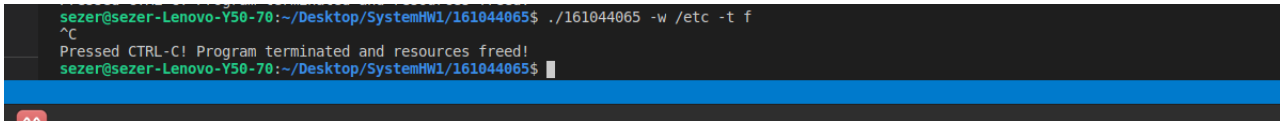

Figure 3: Example of interruption by CTRL-C

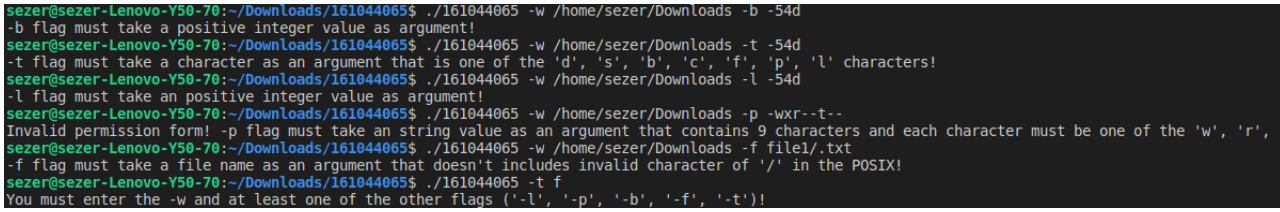

Figure 4: Case of wrong entered arguments

Note:That tex file compiled with pdfLaTeX compiler. Regular LaTeX compiler doesn't recognize some of the functions.