Document Version: 1.0 (a newer version number means an update on the project)
Assignment Date: 1/17/2022, Noon
Due Date - No Penalty: 1/31/2022, 11:59pm
Due Date - 10% Penalty: 2/7/2022, 11:59pm
Due Date - 20% Penalty: 2/14/2022, 11:59pm
No submission is accepted after 2/14/2022, 11:59pm
Group Info: Individual assignment - no groups

Objectives:

- Warm up with C programming
- Practice with data structures, dynamic memory allocation, and pointers
- Practice with systems calls (i.e. related library functions).
- Exercise reading the man pages in Unix
- Trace the systems calls that a program executes.
- Discipline your programs for black-box testing where the expected output will have quite rigit formats.

1 Project Description

In this project, you will write a system program called **`dirlist`**, which takes the absolute path of a directory name (such as `/home/naltipar/Downloads/final-src`) as well as an output file name (such as `output.txt`) as command line arguments, and prints *level*, *order*, and *absolute path* of all the files and sub-directories residing in the given directory into the output file, **level by level, where within each level, order is determined alphabetically by the path name**. For example, assume the given directory path is `/home/naltipar/Downloads/final-src` and the content of this directory is:

```
/home/naltipar/Downloads/final-src
    README.txt
    chap3
        Simulator
            dup.c
        win32-pipe-parent.c
    chap4
        thrd-posix.c
        Driver.java
        thrd-win32.c
```

In the example above, the given directory (`final-src`) has two sub-directories (`chap3`, `chap4`) and a file (`README.txt`). `chap3` has a sub-directory (`Simulator`) and a file (`win32-pipe-parent.c`). `Simulator` has a file (`dup.c`). `chap4` has three files (`thrd-posix.c`, `Driver.java`, and `thrd-win32.c`).

For the given directory path `/home/naltipar/Downloads/final-src`, your program will be invoked as:

```
./dirlist /home/naltipar/Downloads/final-src output.txt
```

and the content of the `output.txt` will be as follows:

```
1:1:/home/naltipar/Downloads/final-src
2:1:/home/naltipar/Downloads/final-src/README.txt
2:2:/home/naltipar/Downloads/final-src/chap3
2:3:/home/naltipar/Downloads/final-src/chap4
3:1:/home/naltipar/Downloads/final-src/chap3/Simulator
3:2:/home/naltipar/Downloads/final-src/chap3/win32-pipe-parent.c
3:3:/home/naltipar/Downloads/final-src/chap4/Driver.java
3:4:/home/naltipar/Downloads/final-src/chap4/thrd-posix.c
3:5:/home/naltipar/Downloads/final-src/chap4/thrd-win32.c
4:1:/home/naltipar/Downloads/final-src/chap3/Simulator/dup.c
```

To achieve this, your program will traverse the disk structure starting from the given directory path in a recursive manner, and build **a linked list data structure** including the `path` and the `level` information of that path. Therefore, each node of the list will be a `struct` having fields to store the path itself (`char *`) and its level (`int`). The linked list will be a ***doubly linked list***, where nodes will have both next and previous pointers.

You are going to build this linked list in sorted order **on the fly** by inserting new elements in ascending (alphabetical) order based on the *path* field, where `strcmp()` will be used for comparison. **This will give you a linked list which is sorted by the path, where levels are mixed but within each level, orders are alphabetically sorted**, such as the following order for the example above:

```
/home/naltipar/Downloads/final-src (level 1)
/home/naltipar/Downloads/final-src/README.txt (level 2)
/home/naltipar/Downloads/final-src/chap3 (level 2)
/home/naltipar/Downloads/final-src/chap3/Simulator (level 3)
/home/naltipar/Downloads/final-src/chap3/Simulator/dup.c (level 4)
/home/naltipar/Downloads/final-src/chap3/win32-pipe-parent.c (level 3)
/home/naltipar/Downloads/final-src/chap4 (level 2)
/home/naltipar/Downloads/final-src/chap4/Driver.java (level 3)
/home/naltipar/Downloads/final-src/chap4/thrd-posix.c (level 3)
/home/naltipar/Downloads/final-src/chap4/thrd-win32.c (level 3)
```

Next, you will design a sorting algorithm to sort the resulting linked list by the `level` field without affecting the order of the nodes within the same level, which are already in sorted order as shown above. After this custom sorting, you will end up with a linked list which is sorted level by level, where each level is also sorted alphabetically, as required by this assignment. Finally, you will print this linked list in the specified format (level:order:path) into the specified output file.

Please check the `readdir()` function and the `dirent` structure to read the content of a directory, and the `stat()` function to differentiate files and directories.

2 Development

It is the requirement of this assignment that you have to use the specified data structures (only one linked list, no queues or trees), and dynamic memory allocation. You cannot assume maximum number of levels, elements, or characters, they can be any size and the memory for them should be allocated dynamically. This will allow you to practice with malloc, pointers and structures.

You will develop your program in a Unix environment using the C programming language. You can develop your program using a text editor (emacs, vi, gedit etc.) or an Integrated Development Environment available in Linux. *gcc* will be used as the compiler. You will be provided a Makefile and your program should compile without any errors/warnings using this Makefile. Black-box testing will be applied and your program's output will be compared to the correct output. A simple black-box testing script will be provided to you for your own test; make sure that your program produces the success message in the provided test. A more complicated test (possibly more then one test) might be applied to grade your program. Submissions not following the specified rules here will be penalized.

3 Tracing System Calls

After finishing your program, you will trace the execution of your code to find out all the system calls that your program makes. To do this, you will use the `strace` command available in Linux. See `man strace` for more details on `strace`. In a separate README file, you will write **only the names** of the system calls that your program made in ascending sorted order by eliminating duplicate names, if any.

4 Checking Memory Leaks

You will need to make dynamic memory allocation. If you do not deallocate the memory that you allocated previously using **`free()`**, it means that your program has memory leaks. To receive full credit, your program should be memory-leak free. You can use `valgrind` to check the memory-leaks in your program. `valgrind` will output:

"All heap blocks were freed - no leaks are possible"

if your program is memory-leak free.

5 Submission

Submission will be done through Blackboard strictly following the instructions below. Your work will be penalized 5 points out of 100 if the submission instructions are not followed. In addition, memory leaks will be penalized with a total of 5 points without depending on the amount of the leak. Similarly, compilation warnings will also be penalized with a total of 5 points without depending on the type or the number of warnings. You can check the compilation warnings using the **-Wall** flag of gcc.

5.1 What to Submit

1. `README.txt`: It should include your name, ID, and the list of system calls that your program made. **Only the unique system call names should be written in ascending order, where each line has a single system call name. No other details about the system calls should be reported.**
2. `dirlist.c`: Source code of your program.
3. `Makefile`: Makefile used to compile your program. If different than the provided one, then you should inform the TA about your changes.

5.2 How to Submit

1. Create a directory and name it as the project number combined with your UofL ID number. For example, if a student's ID is 1234567, then the name of the directory will be ProjectX-1234567 for project X. If it is a group project, then use only one of the group member's ID and we will get the other student's ID from the README file.
2. Put all the files to be submitted (only the ones asked in the *What to Submit* section above) inside this directory.
3. Zip the directory. As a result, you will have the file ProjectX-1234567.zip.
4. Upload the file ProjectX-1234567.zip to Blackboard using the "Attach File" option. You do not need to write anything in the "Submission" and "Comments" sections.
5. **Your project will be graded only once!** If you make multiple attempts of submission before the deadline, then only your latest attempt will be graded. If your submission is late, then it will be graded immediately and you won't be allowed to resubmit once your project is graded!

6 Grading

Grading of your program will be done by an automated process. Your programs will also be passed through a copy-checker program which will examine the source codes if they are original or copied. We will also examine your source file(s) manually to check if you followed the specified implementation rules, if any.

7 Changes

- No changes.