CPTS 360: Lab Assignment 1

Designing a Unix Filesystem Tree Simulator

Notes:

- This is an individual lab.
- Read the entire document carefully before you start working on the lab.
- The code and other answers you submit MUST be entirely your own work, and you are bound by the WSU Academic Integrity Policy (`https://www.communitystandards.wsu.edu/policies-and-reporting/academic-integrity-policy/`).
- You MAY consult with other students about the conceptualization of the tasks and the meaning of the questions, but you MUST NOT look at any part of someone else's solution or collaborate with anyone.
- You may consult published references or search online archives, provided that you appropriately cite them in your reports/programs.
- The use of artificial intelligence (AI)–generated texts/code-snipptes must be CLEARLY DISCOLSED including the prompt used to generate the results and the corresponding outputs the tool(s) provide.

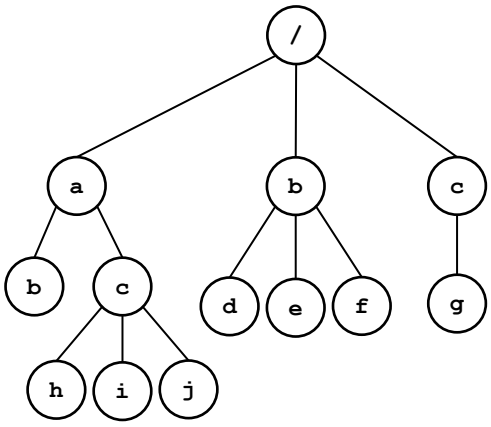
Good Luck!

1 Introduction

The objective of this lab is to get yourself familiar with C programming (including pointers, linked lists, and trees). You will do this by developing a Unix file system simulator.

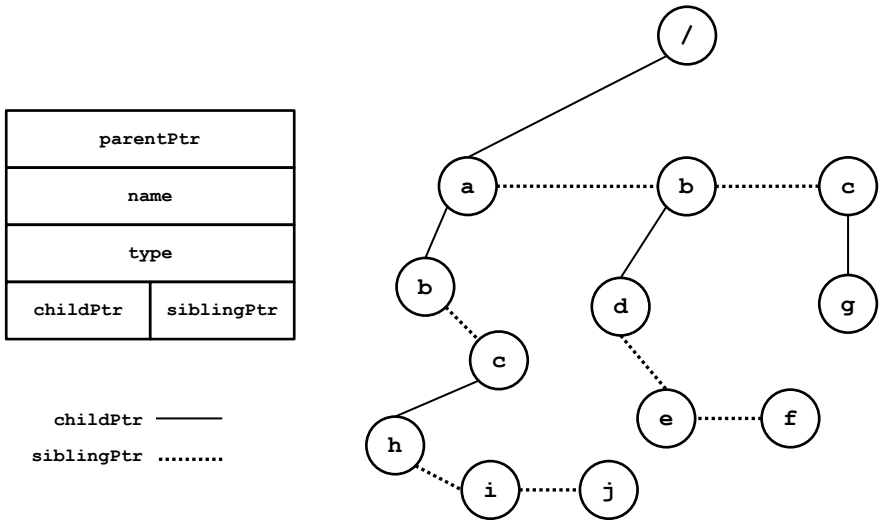
2 Background

The logical organization of a Unix filesystem is a general tree, as shown by the following diagram.



For simplicity, we assume that the tree contains only directories (`DIR`s) and regular files (`FILE`s). The "root" directory is represented by the symbol `/`. A `DIR` node may have a variable number of children nodes. Children nodes of the same parent are called *siblings*. In a Unix filesystem, each node is represented by a unique pathname of the form `/a/b/c` or `a/b/c`. A pathname is *absolute* if it begins with `/`, indicating that it starts from the root. Otherwise, it is *relative* to the Current Working Directory (CWD).

A general tree for our filesystem simulator can be implemented as a binary tree as shown in the figure.



For each node, `childPtr` points to the oldest child and `siblingPtr` points to the oldest sibling. Each node also has a `parentPtr` pointing to its parent node. The `name` and `type` fields used to record the name of the node and corresponding type (either `DIR` or `FILE`). For the root node, both `parentPtr` and `siblingPtr` point to itself.

3 Lab Specifics

In this lab, you will develop a C program to simulate the Unix filesystem. Your program should work as follows.

1. Start with the root node (`/`).
2. Prompt the user for a command (see below for details):

   `mkdir`, `rmdir`, `cd`, `ls`, `pwd`, `creat`, `rm`, `save`, `reload`, `quit`

   (a) Display an error message (`Command not found!`) if the input does not match the above.
3. Execute the command, with appropriate error messages (if the command execution failed).
4. Repeat Step 2 until the `quit` command.

3.1 Command Specifications

- `mkdir pathname`
  Make a new directory for a given `pathname`.
  Show error message (`DIR pathname already exists!`) if the directory already present in the filesystem.

- `rmdir pathname`
  Remove the directory, if it is empty.
  Show error messages if:
  − The directory specified in `pathname` does not exist (`DIR pathname does not exist!`)
  − The directory is not empty (`Cannot remove DIR pathname (not empty)!`).

- `cd [pathname]`
  Change CWD to `pathname`, or to `/` if no `pathname` specified.
  Display an error message (`No such file or directory: pathname`) for an invalid `pathname`.

- `ls [pathname]`
  List the directory contents of `pathname` or CWD (if `pathname` not specified).
  Display an error message (`No such file or directory: pathname`) for an invalid `pathname`.

- `pwd`
  Print the (absolute) `pathname` of CWD.

- `creat pathname`
  Create a new `FILE` node.
  Show error message (`pathname` already exists!) if the directory already present in the filesystem.

- `rm pathname`
  Remove the `FILE` node specified by `pathname`.
  Display an error message (`File pathname does not exist!`) if there no such file exists.
  Display an error message (`Cannot remove pathname (not a FILE)!`) if `pathname` is not a `FILE`.

- `save filename`
  Save the current filesystem tree in the file `filename`.

- `reload filename`
  Re-initalize the filesystem tree from the file `filename`.

- `quit`
  Save the filesystem tree in filename `fssim_lastname.txt`, then terminate the program. The "`lastname`" is your surname.

3.2 Lab Tasks

We provide a skeleton C code (`lab1_base.c`). You will implement additional modules to make the code functional.

4 Useful Hints

4.1 The Tree Node

You may create a C structure `node` with the following entries:

```
struct node {
  char  name[64];
  char  type;
  struct node *child, *sibling, *parent;
};
```

- The `chars` represents "name" string of the node.
- The `type` field denotes the node type: `D` (for directoris) or `F` (for files).
- The `node` pointers `*child`, `*sibling`, and `*parent` points to the appropriate entries in the tree.

4.2 Finding a User Command

We provide a global variable `cmd` that stores all commands:

```
char *cmd[] = {"mkdir", "rmdir", "ls", "cd", "pwd", "creat", "rm",
               "reload", "save", "quit", 0};
```

The following function can find the corresponding index of a given user command.

```
int find_commad(char *user_command)
{
  int i = 0;
  while(cmd[i]){
    if (strcmp(user_command, cmd[i])==0)
    return i;
    i++;
  }
  return -1;
}
```

We can call this function as follows: `int index = find_commannd(user_string)`.

For example, `user_string` value `rmdir` result in `index` = 1.

4.3 Create a New Directory

1. Break up `pathname` into two components, i.e., `dirname` and `basename`, as follows:
   - ABSOLUTE `pathname=/a/b/c/d`. Then `dirname=/a/b/c`, `basename=d`
   - RELATIVE `pathname= a/b/c/d`. Then `dirname=a/b/c`, `basename=d`

2. Search for the `dirname` node:
   - ASSOLUTE `pathname`: search from `/`
   - RELATIVE `pathname`: search from CWD
     - If entry does not exist: show error message and return
     - If entry exists but not `DIR`: show error message and return
3. `dirname` exists and is a `DIR` type:
   - Search for `basename` in (under) the `dirname` node:
     - If an entry already exists: show error message and return
   - ADD a new `DIR` node under `dirname`

4.4 Remove a Directory

1. If `pathname` is absolute, start from `/` (say variable `start = /`)
   Otherwise, start from CWD (i.e., current `DIR` node)
2. Search for `pathname` node:
   - Tokenize `pathname` into components
   - Beginning from `start`, search for each component
   - Return error if fails
3. When `pathname` exists:
   - Check if it is a `DIR` type
   - Check if `DIR` is empty (recall: we cannot remove directory if it is NOT empty)
     - Generate error messages accordingly
4. Delete node from parent's child list

4.5 Create a New File

Similar to create new directories (Section 4.3), except that the node type is `FILE` (i.e., `F`).

4.6 Remove a File

Same as removing a directory (Section 4.4), except check whether the entry is a `FILE`. Besides, we do not need to check whether `pathname` is empty or not.

4.7 Change the Directory

1. Find `pathname` node
   - Generate error message for invalid `pathname` input
2. Check if it is a `DIR`
3. Change CWD to point at `DIR`

4.8 List Directory Entries

1. Find `pathname` node
   - Generate error message for invalid `pathname` input
2. List (print) all children nodes in the form:

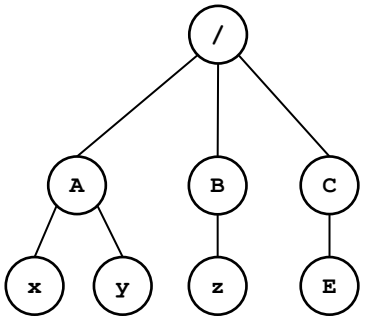
```
TYPE     NAME
TYPE     NAME
TYPE     NAME
....     ....
```

4.9 Display Present Working Directory

1. Find out CWD.
2. Write a helper function that recursively prints CWD from root.
3. Call the helper function from your `pwd` routine.

4.10 Save/Reload Filesystem Tree

Let us assume the filesystem tree is represented by the following:



where `A`, `B`, `C`, and `E` are `DIR`s and `x`, `y`, and `z` are `FILE`s.

The tree can be represented by the (text) lines:

```
TYPE       PATH
----- --------------
 D          /
 D          /A
 F          /A/x
 F          /A/y
 D          /B
 F          /B/z
 D          /C
 D          /C/E
```

You can save the file `filename` as follows:

```
FILE *fp = fopen("filename", "w+");     // open a file stream
fprintf(fp, "%c %s", type, path);       // print a line to file
fclose(fp);                             // close file stream when done
```

You may need to repeat the process to write each line separately.

For reloading the filesystem from saved file `filename`, read each line from `filename` and re-create the filesystem tree.

5 Deliverable

- Complete the implementation of the C code. You can use multiple source/header (`.c` and `.h`) files. However, main file of our implementation (i.e., where the `main()` routine begins) should be `lab1_lastname.c`, where `lastname` is your surname.

  [90 Points]

  - The point split for your implementation is as follows: **80 points** for correctness, **8 points** for good code organization, indentation, and proper comments, and **2 points** for correct C filename.

- A PDF document containing screenshots of your simulator inputs/outputs (see below for details).

  [10 Points]

  - Your PDF filename should be `lab1_lastname.pdf`, where `lastname` is your surname. Include your **full name** and **WSU ID** inside the report. We will **deduct 5 points** if the filename/contents does not follow this convention.

Requirements for PDF Report Containing Screenshots

Show the screenshot of the following commands, run in a sequence:

Sequence	Input	Expected Output
1.	`pwd`	`/`
2.	`rmdir baddir`	`DIR baddir does not exist!`
3.	`ls`	*/* Display nothing as the root directory (`/`) is empty. */*
4.	`mkdir hello`	*/* Create* `DIR hello`. *Display nothing */*
5.	`ls`	`D hello`
6.	`cd hello`	*/* Go to* `DIR hello` *Display nothing */*
7.	`creat hello.txt`	*/* Create a* `FILE` *node (*`hello.txt`*). Display nothing /**
8.	`ls`	`F hello.txt`
9.	`cd`	*/* Change CDW to root (`/`). Display nothing */*
10.	`ls`	`D hello`
11.	`rmdir hello.txt`	`DIR hello.txt does not exist!`
12.	`mkdir hello`	`DIR hello already exists!`
13.	`cd hello`	*/* Move to* `DIR hello`. *Display nothing. */*
14.	`mkdir world`	*/* Create directory world */*
15.	`ls`	`F hello.txt` `D world`
16.	`rm world`	`Cannot remove world (not a FILE)!`
17.	`rmdir world`	*/* Remove* `DIR world`. *Display nothing. */*
18.	`ls`	`F hello.txt`
19.	`cd ..`	*/* Move one level up (CDW is now `/`). Display nothing */*
20.	`rmdir hello`	`Cannot remove DIR hello (not empty)!`
21.	`create hello.txt`	`Command not found!`
21.	`quit`	*/* Save the filesystem with filename* `fssim_lastname.txt` *Display nothing */*

Note: You may use this command sequence to check the correctness and expected behavior of your implementation. We will use the above sequence (and few others) to grade your code.

6 Submission Guidelines

Commit and push your work on GitHub Classroom Lab 1 repository. Get the Git commit id of your work by running the command: `git log -1 -- format=oneline`. Paste the commit id (the hexadecimal string) to the corresponding lab assignment section of Canvas. Check the course website for additional details.

Note: On Canvas, you can submit any commit id (not necessarily the last one) from your Github Classroom Git history. You can also submit as many times as you want. However, we will grade the last commit id submitted to Canvas before the deadline.