COP5570 Programming Assignment 1 (Warm up)
A Global File Organizer

OBJECTIVES

- Get familiar with the UNIX programming environment
- Experience with distributed systems
- Use the `make` utility

DESCRIPTION

In this project, you will develop a global file organizer that organizes files on different machines into a global file system. Using the organizer, the files distributed over different machines can be accessed and operated on using the syntax of conventional UNIX commands. The commands to be supported include

- the `cp` command that copies files in the global file system (files may be moved between machines)
- the `cat` command that displays a file in the global file system
- the `mkdir` command that makes a new directory
- the `cd` command to change the current working directory
- the `rm` command to remove a file
- the `ls` command to list the content of a directory
- the `pwd` command to display the current working directory
- the `mount` command to mount a local directory to the global file system
- the `listmounts` command to display all of the mounted directories
- the `quit` commands to exit the program.

More detailed description follows.

1. The global file organizer organizes files on different machines in one global file system. A local file or directory on a machine is represented as `username@machine_name:path`. An example of such a local directory is `xyuan@linprog1.cs.fsu.edu:/home/faculty/xyuan/tmp`, which represents the directory /home/faculty/xyuan/tmp that can be accessed by user xyuan on linprog1.cs.fsu.edu. In the file organizer, such local directories are mounted to directories in the global file system; and the files under the local directories are accessed using the path in the global file system (global path).
2. When the file organizer starts, it takes a configuration file as a command line argument. Each line of the configuration file has two components: a local directory and a mount directory in the global file system. The following is an example configuration file:

```
xyuan@linprog1.cs.fsu.edu:/home/faculty/xyuan/tmp /linprog1
xyuan@linprog2.cs.fsu.edu:/tmp/xyuan /aaa/bbb/ccc
xyuan@linprog3.cs.fsu.edu:/tmp/xyuan /linprog3
```

With this configuration file, local directory `xyuan@linprog1.cs.fsu.edu:/home/faculty/xyuan/tmp` is mounted on `/linprog1` in the global file system; `xyuan@linprog2.cs.fsu.edu:/tmp/xyuan` is mounted on `/aaa/bbb/ccc`; and `xyuan@linprog3.cs.fsu.edu:/tmp/xyuan` is mounted on `/linprog3`. Hence, the global path `/aaa/bbb/ccc/subdir/xyz` refers to the file or directory /tmp/xyuan/subdir/xyz on linprog2.

To simplify the system design, we will assume that mount directories are all absolate paths (starting with a '/'), and that no local directories are mounted under the mount directory for another local directory.

3. The command './a.out config_file' starts the file organizer. The local directories in the configuration file should be mounted on the global file system. You should maintain the `current working directory` of the file organizer. At the program start, the current working directory is set to the first entry in the configuration file (the default working directory).

4. The file organizer supports the following commands. All commands should support absolute and relative path (relative to the current working directory) for specifying files.

   - **cp src_file dst_file**: Both `src_file` and `dst_file` are global paths (absolute or relative). For example, using the example configuration file, 'cp /aaa/bbb/ccc/subdir/xyz /linprog3' will copy the file /tmp/xyuan/subdir/xyz on linprog2 to /tmp/xyuan on linprog3. For implementation, the command should be translated into the following underlying command:
   'ssh -q xyuan@linprog2.cs.fsu.edu scp -q /tmp/xyuan/subdir/xyz xyuan@linprog3.cs.fsu.edu:/tmp/xyuan'.

   - **cat filepath**: Display the file. Here, `filepath` is a global path (absolute or relative).
   - **rm filepath**: Remove the file. Here, `filepath` is a global path (absolute or relative).
   - **mkdir dir**: Make a new directory. Here, `dir` is a global path (absolute or relative).
   - **cd [dir]**: Change the current working directory. Here, `dir` is an optional global path (absolute or relative). If `dir` is empty (the command is just 'cd'), the current working directory is changed to the default current working directory (first entry in the configuration file).
   - **ls [dir]**: Display the content in a directory. Here, `dir` is an optional global path (absolute or relative). If `dir` is empty (the command is 'ls'), the content of the current working directory should be displayed.
   - **mount local_directory dir**: Mount the `local_directory` to the global path `dir`. Here, `dir` can only be an absolute path while `local_directory` is of the form `username@machine_name:path`.
   - **pwd**: Display the current working directory.
   - **listmounts** Display all mount directories and their associated local directories.
   - **quit**: Quit the program.

5. For error checking, your program only needs to perform two basic checks, (1) command not supported and (2) global path in the command is not mounted. You can assume all other inputs are correct.

6. You can try the sample executable to see how the program should behave.

GRADING POLICY

This program should be developed on linprog. A proper makefile should be used in this project: (1) Command 'make' should produce the executable; (2) Command 'make demo' should run the program (use the existing executable if it has been generated or produce the executable if it is not available); (3) Command 'make clean' should remove .o files and executables from the directory; and (4) the program should be compiled with '-Wall -ansi -pedantic' or '-Wall -std=XXX -pedantic' flags. Compiling the program with flags should not have any warning message.

You should also write a README file that includes the basic README file header given in the website and gives information about (1) how to compile and run your program and (2) how to repeat your demo (you will design a demo for your assignment).

- A program that does not compile gets 0 point. A program that does not run gets 0 point. A program that does not support any specified commands get 0 point.
- Proper makefile and README file(10 pts). A makefile that does not compile the program with the '-Wall -ansi -pedantic' or '-Wall -std=XXX -pedantic' flags will lose all 10 points.
- mount the directories in the configuration file (5 pts)
- 'cp' (5 pts)
- 'cat' (5 pts)
- 'mkdir' (5 pts)
- 'ls' (5 pts)
- 'rm' (5 pts)
- 'mount' (5 pts)
- 'cd' (5 pts)
- 'pwd' (5 pts)
- 'listmounts' (5 pts)
- 'quit' (5 pts)
- error checking (5 pts)
- Demo (10 pts). You must design your demo to show all features in your program WITHIN THE GIVEN TIME (15 minutes). Failing to do so will lose not only the demo points, but also the points for the features that you do not have time to demo. A demo should always start from downloading your submission from canvas. You will not be allowed to modify your source code (unless you are asked to) after the submission deadline. One will lose all 10 points if the source code needs to be modified during demo.
- **Extra 15 points deduction for the first unknown bug of any kind. Extra 8 points deduction for the first known bug or unimplemented feature**
- 5 points deduction for each warning message in the compilation.
- Reporting a bug in the sample executable gets extra 5 pts.

DEADLINES AND MATERIALS TO BE HANDED IN

January 24, 11:59pm. Submit the following files in a tar file to canvas

- All project related files including makefile, README, and self-grading file. In a directory that contains your submitted files, type 'make' should compile the source C files with proper flags and produce the executable for the project.
- The self-grading sheet with the grades you give to your project.

The project demo will take place in the week after the due date.

MISCELLANEOUS

- All programs will be checked by an automatic software plagiarism detection tool.
- This is a small program, about 200 lines of C code in the sample implementation.