Pesky Permissions ‘n suPerusers

This assignment is an extension of the Fabricated Files ‘n Folders homework assignment. Feel free to use the provided key for Fabricated Files ‘n Folders as a starting point.

The goal is to now make the permissions you previously set with “chmod” actually work! At this point, things like users and groups should NOT be arbitrarily assigned (the current user should own what is created).

**CONSOLIDATED INSTRUCTIONS**

When implemented, these commands should complement the core functionality of the file system commands you have already implemented. When users do not have read permissions, the files/directories should no longer appear. When users do not have write permissions, they should not be able to rm/rmdir the files. When users do not have execute permissions they should not be able to ./ the file. Note: ./ functionality can be as easy as printing “[file\_name] ran/executed.” For each of the RWX permissions, the most permissive available should be used. In other words, if the group permissions prohibited reading but the owner permissions gave read permissions, the user would be allowed to read. Likewise, if a user was part of the group where group permissions allowed reading but was not the owner and global reading was not allowed, the files/folders should still be visible (by being in the group that held the permissions).

You can check Linux Man pages for the usage or see the examples in the detailed discussion for each command. The required commands to support in this iteration are: useradd, useradd -G, usermod -a -G, chown, chgrp, ./ (execute), groups, groupadd, userdel, userdel -G

In addition to the standard Linux-based commands, we need a way to switch users. Implement a new switchto command which will switch to a different user.

This program is to be coded in C++ and submitted in Canvas before the deadline. This is an individual assignment – each person should produce and submit their own code solution.

If you want to better understand the permissions structure in Linux, you need to make a slight distinction between permissions of files and those of folders. The nitty gritty details on how things are handled are fairly concise at <https://www.thegeekstuff.com/2010/04/unix-file-and-directory-permissions/>

**Bonus** – The baseline commands should work for a single group at a time (see useradd for example). If you want to allow for multiple groups as the real command does, that could qualify for a bonus.

The su/sudo commands can also allow switching of users. Typically to allow execution in this manner requires being part of a group allowed to sudo (or knowing the root password for su). Adding additional Linux functionality with a subset of these commands could qualify for a bonus. Also coupling with the passwd command (to temporarily setup passwords) could make for an interesting bonus!

**DETAILED INSTRUCTIONS**

First we need some groups (you can’t arbitrarily assign yourself to groups which don’t exist!).

groupadd <group>

a typical default group is “users” (all new users get automatically added to the users group)

Users can be members of groups (and could be members of many groups), so there have to be commands to make a member part of a group (or removed from group membership).

Adding an existing user to a group is done with usermod –a –G <group> (works so long as group exists)

Creating a new user and adding groups can be done with useradd –G <group[,<group>]> <username>

Or a user can be added into the system with just the default group using useradd <username>

The basic setup for a root user (adding to existing groups called root, users, and wheel) would look something like this (notice NO whitespace between groups): useradd –G root,users,wheel root

Let’s allow removal of users from groups with userdel –G <group> <username> and while we’re at it, be able to remove users from the system with userdel <username>

What groups is a user in? Use groups <username> to print out the groups the user is a member of

Need to change the owner or group? chown <owner> <file> will change ownership and chgrp <group> <file> can change the group. If you want to emulate Linux, chown could optionally set both owner and group with a semicolon in between owner and groups as in chown <owner>:<group> <file>

We need something to represent being able to “run” files with the execute permissions. The standard ./<file> can work as previously mentioned

Finally let’s include an easy way to switch between these users with a new made-up command switchto <username>. There are no restrictions on this command, it immediately switches to the new user. If you want to make it a little easier to keep track of who is the current user, add the whoami command like Linux provides.