Breakout / Lab 09

Change the Mode

CSCI 1730 – Fall 2015

Problem / Exercise

For this breakout lab, you are going to implement the `chmod` utility from scratch. You are NOT allowed to use the following system calls in your implementation: `fork`, `exec`, and `system` (or related functions). You are allowed to use `chmod`, `fchmod`, `fchmodat` and other system calls (but not the existing `chmod` utility itself). Here is the manual page for the `chmod` utility you are going to implement:

```
NAME
       chmod - change file mode bits

SYNOPSIS
       chmod -sym MODE[,MODE]... FILE
       chmod -oct OCTAL-MODE FILE

DESCRIPTION
       This  manual  page  documents the  CSCI 1730 version of chmod.  chmod changes the file mode bits
       of  each  given file according to mode, which can be either a symbolic representation of changes
       to make, or an octal number representing the bit pattern for the new mode bits.

       The  format  of a symbolic mode is [ugoa...][[+-=][perms...]...],  where perms is either zero or
       more letters from the set rwxXst, or a single letter from the set ugo.   Multiple symbolic modes
       can be  given,  separated by commas.  This format is the same as in the GNU version of the chmod
       utility.

       A numeric mode is from one to four octal digits (0-7), derived by adding up the bits with values
       4, 2, and 1.  Omitted  digits  are assumed  to be  leading zeros.   The  first digit selects the
       set user  ID  (4)  and  set  group ID (2) and restricted deletion or sticky (1) attributes.  The
       second  digit  selects  permissions  for the user who owns the file:  read (4),  write (2),  and
       execute (1);  the  third selects permissions for other users in  the files group, with the  same
       values; and the fourth for other users not in the files group, with the same values.  This octal
       format is the same as in the GNU version of the chmod utility.

       chmod never changes the permissions of symbolic links; the chmod system call cannot change their
       permissions.  This  is  not  a  problem since  the permissions of symbolic links are never used.
       However, for each symbolic link listed on the command line, chmod changes the permissions of the
       pointed-to file.  In  contrast,  chmod  ignores  symbolic  links  encountered  during  recursive
       directory traversals.
```

Notes

- **User Input:** You may **NOT** assume valid user input.
- **Error Handling:** If a system call results in an error, then your implementation should display the error using `perror` (available in `⟨cstdio⟩`). In general, if an error occurs (e.g., invalid input), then display a message to the user indicating what the error is and exit with status `EXIT_FAILURE` (available in `⟨cstdlib⟩`).
- This lab is designed as a lead-in to Project 2 where you will be implementing a collection of Unix utilities. Keep this in mind while implementing your lab code. For example, if you write a function that you think might be useful in other utilities, you might move it (or a collection of functions) into a separate `.cpp` file and include its prototype in a header file so that it can be used by other utilities that you write.

1 Things to Consider

Here is a list of things to consider:

- What system calls do you need in order to solve the problem?
- How are you going to check for valid user input?
- What potential errors can occur?
- How are you going to parse user input?

You may find the following useful for parsing command-line options: `http://www.gnu.org/software/libc/manual/html_node/Getopt.html#Getopt`.

2 Group Brainstorm

You are NOT allowed to use the computers during this time.

Breakup into groups based on your seating and brainstorm about how to solve the problem or exercise. Make sure everyone understands the problem, and sketch out potential ways to move towards a solution. Perhaps something that was discussed during lecture might be useful?

NOTE: For this breakout lab, determine what each group member already knows about the topics enumerated at the beginning of the "Problem / Exercise" section. Also talk about what you expect to learn from this breakout lab.

3 Submit Individual Brainstorm

You may use a computer from this point forward.

Login to eLC and submit a version of your group's brainstorm, written in your own words. You may add additional information if you want. You need to write enough in order to convince the grader that you understand the problem or exercise and that you have a plan for moving forward towards a solution. Please include the last names of the other people in your group in your submission. The brainstorm submission should be available on eLC in your assignment dropbox. We prefer that you submit your individual brainstorms before the end of your breakout period, however, you generally have until 3PM on the day of your breakout (as indicated on eLC) to submit them.

NOTE: Submissions that do not include an individual brainstorm will not be graded. Also, once you have completed your individual brainstorm, please use the remainder of the lab period to actually work on the lab. Failure to do this may result in the taking of attendance.

4 C++ Code

Make sure that all of your files are in a directory called `LastName-FirstName-lab09`, where `LastName` and `FirstName` are replaced with your actual last and first names, respectively. The code for this lab should be written in a file called `chmod.cpp`.

4.1 `Makefile` File

You need to include a `Makefile`. Your `Makefile` needs to compile and link separately. Make sure that your `.cpp` files compile to individual `.o` files. The resulting executable should be called `chmod`.

4.2 `README` File

Make sure to include a `README` file that includes the following information presented in a reasonably formatted way:

- Your Name and 810/811#
- Instructions on how to compile and run your program.
- A Reflection Section. In a paragraph or two, compare and contrast what you actually did to complete the problem or exercise versus what you wrote in your initial brainstorm. How will this experience impact future planning/brainstorms?

NOTE: Try to make sure that each line in your `README` file does not exceed 80 characters. Do not assume line-wrapping. Please manually insert a line break if a line exceeds 80 characters.

4.3 Compiler Warnings

Since you should be compiling with both the `-Wall` and `pedantic-error` options, your code is expected to compile without `g++` issuing any warnings. For this lab, compiling without warnings will be one or more of the test cases.

4.4 Memory Leaks

You are expected to ensure that your implementation does not result in any memory leaks. We will test for memory leaks using the `valgrind` utility. For this lab, having no memory leaks will be one or more of the test cases.

5 Submission

Before the day of the next breakout session, you need to submit your code. You will still be submitting your project via nike. Make sure your work is on `nike.cs.uga.edu` in a directory called `LastName-FirstName-lab09`. From within the parent directory, execute the following command:

```
$ submit LastName-FirstName-lab09 cs1730a
```

It is also a good idea to email a copy to yourself. To do this, simply execute the following command, replacing the email address with your email address:

```
$ tar zcvf LastName-FirstName-lab09.tar.gz LastName-FirstName-lab09
$ mutt -s "lab09" -a LastName-FirstName-lab09.tar.gz -- your@email.com < /dev/null
```