CSCI 330 Assignment 4 (Spring 2021)

Shhhhhhh.... it's a secret! (100 pts)

Purpose

This assignment should give you experience in using file descriptors, `open(2)`, `close(2)`, `write(2)`, `stat(2)` and `chmod(2)`, `perror(3)`, as well as working with command line arguments. The numbers in parentheses here are section numbers so you can find the right manpage.

Program

A typical UNIX system will have many files that contain sensitive information. Permissions can keep these files secure. Some files can be publicly read, but can not be altered by a regular user (eg. `/etc/passwd`). Other files can't be read at all by a regular user (eg. `/etc/shadow`). Your task is to write a C++ program that will allow you to add messages to a file that has **NO** permissions granted for any user.

The program you develop will take a message as a command line argument and append that message to the end of a file (which is also specified as a command line argument). It will also support a `-c` command-line option that, when supplied, will cause your program to clear the file before the message is appended.

The file should have all permissions denied for any user, both before and after the message is appended. For this to work, the person running the program needs to be the owner of the file.

Algorithm

1. Check to see whether the output file exists. If it doesn't, create it. It will make things easier for you if any newly created file is closed at the end of this step.
2. Check the output file's permissions. If any exist, print a useful error message and exit.
3. Change the permissions on the file to allow writing by the user.
4. Open the file for output. If the `-c` command line option is present, *truncate* the file contents.
5. Write the message passed in on the command line into the output file. Write an additional newline character so that the output has a nicer format.
6. Clear the permissions and close the file. (These two operations can be performed in either order, but the implementation will be slightly different.)

Useful Hints

- Don't use the command line arguments directly. Their position in the argument list may change depending on command line options (like your "-c"). Create meaningful `char *` variables and fill these with the appropriate entries from the argument list, once you've determined their proper positions. I would recommend using `getopt(3)` to handle the command line options/arguments.
- This program will not be reading in any kind of input except for the command line arguments. If you're trying to use `cin` or `read` for something, you're probably doing something wrong.

Error Checking

- If the message log file cannot be opened, an appropriate error message should be printed to standard error and the program should exit immediately. If the file has any permissions at all, the file should be rejected as insecure, and the program should exit.

- Check for error values after every system call that can fail (that's most of them), and print out what happened when an error did occur. It is more work to set up, but when something goes wrong, you'll actually have an idea what caused the problem.

Example Run

```
# Note: seclog is the compiled program in executable form
% rm log
% ./seclog
Usage: seclog [-c] out_file message_string
       where the message_string is appended to file out_file.
       The -c option clears the file before the message is appended
% chmod u-w .
% ./seclog log "Hello"
Permission denied
% chmod u+w .
% ./seclog log "Hello 1"
% ls -l
---------- 1 z123456 student     8 Sep 24 18:39 log
-rwxr-xr-x 1 z123456 student 26385 Sep 24 18:38 seclog*
-rw-r--r-- 1 z123456 student  2204 Sep 24 18:36 z123456.cc
-rw-r--r-- 1 z123456 student 30896 Sep 24 18:38 z123456.o
% ./seclog log "Hello 2"
% ls -l
---------- 1 z123456 student    16 Sep 24 18:40 log
-rwxr-xr-x 1 z123456 student 26385 Sep 24 18:38 seclog*
-rw-r--r-- 1 z123456 student  2204 Sep 24 18:36 z123456.cc
-rw-r--r-- 1 z123456 student 30896 Sep 24 18:38 z123456.o
% chmod 400 log
% cat log
Hello 1
Hello 2
% ./seclog log "Wait, there's more"
log is not secure.  Ignoring.
% chmod 000 log
% ./seclog log "Wait, there's more"
% ls -l
---------- 1 z123456 student    35 Sep 24 18:41 log
-rwxr-xr-x 1 z123456 student 26385 Sep 24 18:38 seclog*
-rw-r--r-- 1 z123456 student  2204 Sep 24 18:36 z123456.cc
-rw-r--r-- 1 z123456 student 30896 Sep 24 18:38 z123456.o
% chmod 400 log
% cat log
Hello 1
Hello 2
Wait, there's more
% chmod 000 log
% seclog -c log "Clean start"
% ls -l
total 72
---------- 1 z123456 student    12 Sep 24 18:41 log
-rwxr-xr-x 1 z123456 student 26385 Sep 24 18:38 seclog*
-rw-r--r-- 1 z123456 student  2204 Sep 24 18:36 z123456.cc
-rw-r--r-- 1 z123456 student 30896 Sep 24 18:38 z123456.o
% chmod 400 log
% cat log
Clean start
% chmod 000 log
```

What to turn in?

Submit, through Blackboard, the following:

- A single C++ source file implementing the program. Use the name `a4-z123456.cc`, but use your Zid in the name instead of the example one.

Remember, there is **no credit for late work** and, as usual, the grading will be performed on `turing` and/or `hopper`, so you must ensure that your program compiles and runs properly on them. It does not matter if it was working somewhere else if it doesn't work on those machines where it is finally tested.