Assignment 2 for CS570, Summer 2018

You shall implement a multi-processing file editor. The file editor shall provide the following functions:

1. Create new directory files
2. Create new regular files
3. Create child process to write sorted output
4. Create child process to shadow write/read regular files
5. Read from a file (print out to stdout)
6. Write to a file in either insert, append, or overwrite mode
7. Print file status (print out to stdout)
8. Print directory listing (contents of dir file) (print out to stdout)

Your program shall have the following features built into your project:

- When the program starts up it shall provide the user with a "menu" of choices for creating files (dir & regular file types), reading, writing (offer choice for writing modes) print file status, and print a dir listing. Most of the menu choices require a filename as well, be sure to prompt (ask) the user for a filename.
- Whenever a user selects an option to read from or write to a file, you will create a child process which makes a copy of the file the parent is going to read/write from/to and will update it as the user updates the "real" file. This "shadow" file shall have the same filename as the "real" one except that the name shall be appended with ".bak" (e.g. a file named file.txt will have a shadow file named file.txt.bak). If a file already exists by that name, then delete it and use the name for this file.
- When the user is done editing a file (writing to a file), then two child processes shall be created. One child process shall create a new file and populate it with the contents of the file just written, the contents shall be sorted in alphabetic order. The second one shall do the same but sorted in reverse order.
- All three child processes will then terminate after all of the sorting and the user's menu will be redisplayed. NOTE - all three processes will not terminate until all three have completed their work.
- For editing (writing) a file, the user shall be allowed to write to a file by appending to an existing file, creating a new file (overwriting an old), or inserting into a new file. If inserting into a new file, then the user shall be allowed to specify what byte to begin inserting the new data into.

Your program will be tested by compiling it and executing it on edoras. Your program shall be written such that it compiles and executes cleanly when using the gcc/g++ compiler on edoras. You shall create a sub-directory named "a2" in your home directory. In it, you shall place your source files (multiple source files are required), your header file, your Makefile, and a README file (follow instructions from assignment #1 for the README file). Additionally, identify in your README file who worked on which lines of code in this project (if you used Agile/Pair programming state who was writing and who was providing input for each function/method). Your source files SHALL CONTAIN sufficient comments for making the source easy to read. Points will be taken off for poorly (or non) commented source. Name the executable "filem".

- Create ~/a2 by hand.
- Create multiple c source files, an include file, a Makefile, and a README file. Put them into ~/a2.
- The Makefile shall create an executable by the name of filem in the same directory (~/a2).
- The system call "system()" will NOT be allowed

The assignment is due 1800 (6:00 PM) on Wednesday, 20 June 2018

TURNING IN YOUR WORK:

Your project files shall be loaded onto Assignment #2 on Blackboard and into the class account.

Make sure that all of your files (all source files, Makefile, README file, test files, etc) are in the a1 sub- directory of the class account

Additionally, create a tarball (tar file) or zip file and attach that file (upload it) under Assignment Submission in Assignment #2 in Blackboard.

SUGGESTION: check out my assignment grading policy and sample source and Makefiles posted on Blackboard.

HINTS ON MENU:

- First, display the numbered user options.
- The user should only type a number to choose the correct option which your programs responds to.
- Reserve the number 9 or the number 0 to exit the menu (or sub-menu) and use that number consistently throughout all of your (sub)menus.
- Each option should then display sub-options (where appropriate) and prompt the user for the correct input from the user. Sub menus should always provide the user with the option to return to the previous level menu.