Patrick Welch
CSCE 311 Project 4 Report

Video link: https://youtu.be/1zDYSYJ16Xk

Program Description

There are three programs, orw.c, forw.c, and fmap.c. They each can be run by using ./<name>.out, for example ./orw.out from the directory they're stored in. They each also have the file path being passed through an argument, so one would include the file name when they run the program, ie ./owr.out /this/is/a/file/path. There are two make commands you can run: (1) "make" which will compile all three programs, and (2) "make clean" which will remove all the .out files from the folder.

orw.c uses the open/read/write system calls in order to switch the cases of the contents of a text file. It opens the file twice with two different file descriptors, one for reading and the other for writing. After opening there is a while loop that is used to read chunks of the text file at a time to a buffer, switch the case, and then write it to the file descriptor that has the writing permissions. Once it gets to a point where the number of bytes read is less than the number of bytes in the buffer, that means we've reached the end, so a null terminating character is added to the buffer and then the remaining contents of the file have their case switched. Afterwards, it closes both of the files using the close() call.

forw.c uses the fopen/fread/fwrite functions in order to switch the cases of the contents of a text file. First, it opens a text file using a file pointer named "fptr" which is opened with only reading permissions. Then the fseek() and ftell() functions are used to determine the size of the file, and after that fseek is used again to reset the pointer to the beginning of the file. The next step is that a char* buffer gets space allocated to it the size of the file. Once that happens, the entire file is fread into the buffer, and then fclose is used to close the file pointer. Now the program switches the case of all the contents of the buffer, and then another file pointer is used to fopen the file again, this time with writing permissions, using a file pointer named "ftpw". fwrite is then used to write the entire contents of the buffer into the text file. Finally, the memory allocated to the buffer is freed and the file pointer "fptw" is closed using fclose.

fmap.c uses open/close to gain access and close the file, and mmap to read and write from the file. As opposed to the first program, only one file descriptor is used this time, which has read and write permissions. fstat is then used to get the size of the file. A char* 'p' is then linked to the contents of the file by using mmap. After that, any changes made to 'p' are saved to the file. A for loop then runs through 'p' and switches the case of all the contents. Finally, munmap is used to un-map the contents from p and the file descriptor is closed.

Questions:

1. The read/write APIs are system calls which means it makes direct calls to the kernel. They are also not formatted. The fread/fwrite APIs are functions that are in the standard C library. Uses an internal buffer and is formatted as well. The mmap API maps the file into memory, and can be shared between the file and the program, allowing for essentially real-time editing of the file.
2. A file pointer points to a FILE struct, allowing for the identification and management of files. A file descriptor is an int that is used for identifying a specific opened file at the kernel level (Like how I mentioned earlier that the read/write APIs are system calls). A file description is "an entry in the system-wide table of open files". (open(2), man7.org). Meaning that the file description is the information about the file that the file descriptor refers to, like the file's offset and its status.
3. /proc is a virtual filesystem that holds info about running processes and assists the user and kernel in communicating.
    a. /proc/[pid]/cmdLine: Here, [pid] is simply a placeholder for a process number. This file gives the process access to the command line that was used in running it.
    b. /proc/[pid]/fd: Here as well [pid] is a placeholder for a process number. This file has the file descriptions of all the files that the given process has currently open, named by its file descriptor.
    c. /proc/sys/kernel/sem: This file holds four values that define the limits for the IPC semaphores. The fields are SEMMSL, SEMMNS, SEMOPM, SEMMNI. These are the maximum number of semaphores per semaphore set, the maximum number of semaphores in all semaphore sets through the whole system, the number of operations that can be used in a semop() call, and the maximum number of semaphore identifiers allowed through the whole system (respectively).