CS5600 Homework 0

Assigned 1/8/2018
Due 1/14/2018 at 11pm PST

Your task is to implement a version of the unix tool `cp`, but you must rely only on file descriptors and posix API's to do so. That is, you must not use the file API's that take a `FILE*`, such as fopen or fclose, nor should you use anything from the C++ iostream library. You should not use any external libraries of any kind outside of standard libc. For example, to open a file and get a file descriptor back use `open`:

```
int fd = open("file.txt", O_RDONLY);
```

You should start out by forking the hw0 repo in the cs5600 course github organization:

```
https://github.ccs.neu.edu/cs5600-03-sp2018/hw0
```

Then you must add my git user as a collaborator: `jrafkind`

Put your code in the `src/main.cpp` file.

Your version of `cp` should have the following functionality:

- The program must be named 'cs5600-cp'
- Make a bit for bit copy of the file contents from the source to the destination
- The command line syntax should be

  ```
  $ cs5600-cp <src-file> <dest-file>
  ```

  The first argument names a file and will be copied to the destination named by the second argument. If the second argument is an existing file then print a warning and do not do the copy. If the destination argument is an existing directory then check if there is a file in the given directory whose name is the first argument. If the destination file does not exist then do the copy, otherwise print a warning and do nothing further. Example:

```
# Copy the file from x to b, where b does not already exist
$ cs5600-cp x b
# The second attempt should do nothing because b already exists
$ cs5600-cp x b
Warning: 'b' already exists
```

```
# Copy x to a directory named d. The result will be d/x
$ cs5600-cp x d
$ ls d
a
# A second copy to d will produce a warning
$ cs5600-cp x d
Warning: 'd/x' already exists
```

If less than two arguments are passed then print a help message

```
$ cs5600-cp x
Please give two arguments.
```

If the source is not a file or does not exist then print an error.

```
$ cs5600-cp does-not-exist b
Warning: 'does-not-exist' does not exist
$ cs5600-cp some-dir b
Warning: 'some-dir' is a directory
```

The destination file should have the same file permissions as the source file, as if both files had the same arguments to chmod applied to them.

```
$ ls -l a
-rw-rw--- a
$ cs5600-cp a b
$ ls -l b
-rw-rw--- b
```

You can verify that your copy worked correctly by manually inspecting the contents of the two files is the same, or use a cryptographic hash on the contents and check that the two hashes are the same.

```
$ sha1sum a
7e94486478a414b055d043fc6403a32e2590a264
$ sha1sum b
7e94486478a414b055d043fc6403a32e2590a264
```

Other cryptographic hash tools are `md5sum`, `sha256sum`, `sha384sum`, and `crc32`. For this assignment any tool will work.

- Run `test.py` to check if your program is working correctly.

```
$ python test.py
Copying /tmp/test/tmpWD3DU9 (1502 bytes) to /tmp/test/tmpLtoBlG
passed
```

Things to consider:

- You should always check the return codes of system calls (and library functions in general), and make an attempt at doing something reasonable if an error should occur. Many system calls will return -1 if the call failed, and set the global `errno` value to the actual error condition. You can use the function `perror` to print out a description of the failure, or use `strerror` to get a string representation.

```
int fd = open("file.txt", O_RDONLY);
if (fd == -1){
  perror("open failed:");
  const char * error_str = strerror(errno);
  // either abort the current logic, or return some error condition
}
```

- Use `malloc` and `free` for dynamic memory management. It is not critical that there be no memory leaks in your program, but make a reasonable attempt at cleaning up all resources you use. That also means you should call `close` on any open file descriptors before your program exits. To make sure you have no memory leaks you can use the tool `valgrind`. Simply pass your program as an argument to `valgrind`:

  ```
  $ valgrind cs5600-cp
  ```

  You can learn more about valgrind at http://valgrind.org/

- To get you started, here are a list of functions that you might need. For a better explanation on any function read the corresponding man page. Occasionally there are multiple man pages for a single name, but if you pass 2 to man then you can usually get to the system call manual. Example

  ```
  $ man 2 open
  ```

  The man pages are also available on the internet if you search for them using a search phrase such as `man unix open`

```
open - open a file and return a file descriptor
close - close a file descriptor
read - read data from a file descriptor
```

```
write - write data to a file descriptor
stat, fstat - get information about a file either through its
  name (stat) or file descriptor (fstat)
opendir - open a directory and return a directory handle
readdir - get information about a directory
closedir - close a directory handle
access - checks whether the file can be accessed
```

Style

Please use sensible names for functions and variables. Do not name a number `n`, do not name a string `s`, and do not name a file `f`. Complete words are generally the best choice for names.

If you have a preferred C-style for formatting the syntax then stick to that, otherwise I prefer a style as follows.

```
void foo(int number){
    if (number > 3){
      printf("number is %d\n", number);
    }
}
```

Where the opening `{` is on the same line as the function/if, the indentation is 4 spaces, and there is a space surrounding operators.