Lab 2 - Filesystem System Calls

File Descriptors

- A file descriptor is like a "ticket number" representing your currently-open file.
- It is a unique number assigned by the operating system to refer to that file
- Each program has its own file descriptors
- When you wish to refer to the file (e.g. read from it, write to it) you must provide the file descriptor.
- File descriptors are assigned in ascending order (next FD is lowest unused)

System Calls

- Functions to interact with the operating system are part of a group of functions called system calls.
- A system call is a public function provided by the operating system. They are tasks the operating system can do for us that we can't do ourselves.
- open(), close(), read() and write() are 4 system calls we use to interact with files.

open()

- A function that a program can call to open a file:

```
int open(const char *pathname, int flags);
```

- pathname: the path to the file you wish to open
- flags: a bitwise OR of options specifying the behavior for opening the file the return value is a file descriptor representing the opened file, or -1 on error
- Many possible flags (see man page for full list).
- You must include exactly one of the following flags:
    - O_RDONLY: read only
    - O_WRONLY: write only
    - O_RDWR: read and write
    - Another useful flag is O_TRUNC: if the file exists already, clear it ("truncate it").

close()

- A function that a program can call to close a file when done with it.

```
int close(int fd);
```

- It's important to close files when you are done with them to preserve system resources.
- fd: the file descriptor you'd like to close.
- You can use valgrind to check if you forgot to close any files.

read()

```
// read bytes from an open file
ssize_t read(int fd, void *buf, size_t count);
```

- fd: the file descriptor for the file you'd like to read from
- buf: the memory location where the read-in bytes should be put
- count: the number of bytes you wish to read
- The function returns -1 on error, 0 if at end of file, or nonzero if bytes were read (may not read all bytes you ask it to!)

write()

```
// write bytes to an open file
ssize_t write(int fd, const void *buf, size_t count);
```

- Same as read(), except the function writes the count bytes in buf to the file, and returns the number of bytes written.

Programming Exercises

1. create_file.cpp : This program shows an example of creating a new file with a given name. Similar to the `touch` unix command, it creates the file if it doesn't exist. But in this case, if it does exist, it throws an error.
2. copy.cpp : This program is the starter code for an example that shows how we can make a copy of a specified file into another specified file, similar to the `cp` unix command.
3. copy_extended.cpp : This program is an implementation of an extended copy command that shows how we can make a copy of a specified file into multiple other specified files, as well as printing the contents to the terminal.