Exercise 01

For this exercise, you're going to be comparing system calls and the library routines you're probably more accustomed to using. I've written a short program for you, writeLibc.c. You'll find it on the course homepage in Moodle. This program uses the C standard library to create an output file, output.txt. It uses fputc() to write more than 1,000,000 characters to the file, one character at a time, rewinding to the start of the file periodically, so it doesn't get too large.

Compile this program and time its execution on one of the EOS linux machines. The following commands should do the job:

```
$ gcc -Wall -std=c99 writeLibc.c -o writeLibc
$ time ./writeLibc
```

Once you're done with this program, delete the output file it created. This will help you make sure you don't make mistakes on the next part:

```
$ rm -f output.txt
```

Let's compare this program to one that uses the system call interface directly to write the output file. Copy writeLibc.c to a file named writeSystem.c. I'll summarize what you'll need to do to convert the program to using system calls to write the file, but you'll also need to read some of the documentation yourself. You can read the online documentation for a system call like open() by using the **man** command. The **2** option below says to get the documentation from section 2 of the online manual, where system calls are documented:

```
$ man 2 open
```

- Since you'll be using the system call interface, you'll use a file descriptor (an int) to keep up with the file, rather than a file pointer.
- As you start putting system calls in the program, you will need to include more headers at the top. The online documentation for each system call tells you what headers you'll need to include.
- Replace the call to fopen() with a call to the open() system call. It expects the filename as the first parameter and a bitwise OR of various options as the second parameter. You'll need to bitwise or together options that say you want to open the file for writing and that you want to create the file if it doesn't already exist. There's an optional, third parameter that you'll need to supply since you're creating a file. It lets you specify permissions for the new file. An octal value like 0600 would be a good choice for this; it gives the owner (you) the right to read and write the file.
- Replace the call to fputc() with a call to the write() system call. Write will let you write out as many bytes as you want, given an array of bytes (i.e., a pointer) and the number of bytes it should write. For this exercise, you're just going to write out one byte at a time. You'll need to pass the address of the byte you want to write as the second parameter to write().
- Replace the call to rewind() with a call to lseek(). Using the constant SEEK_SET as the last parameter will make it easy to rewind back to the start of the file.

- Replace the call to fclose() with a call to the close() system call.

When you're done, compile this version of the program and time its execution:

```
$ gcc -Wall -std=c99 writeSystem.c -o writeSystem
$ time ./writeSystem
```

You'll probably see that this version of the program runs much more slowly than the one that uses libc. At first, this might be surprising, since writeSystem makes system calls directly, while writeLibc goes through a library. The difference in performance is because making a system call can be much more expensive than just calling a subroutine. The writeSystem version of the program really has to make more than a million system calls, with a little bit of runtime overhead for each one. In writeLibc, we make a million calls to fputc(), a library function. The libc library still has to eventually make system calls to actually write characters to the file, but libc is smart enough to buffer lots of single-character writes before making a system call to write out multiple characters at once.

If you want to see the difference, you can run both of these programs through strace. This will report on every system call made by either program. You will probably want to kill each of these with ctrl-C before they finish, since they both produce lots of output. Before you kill them, you should get to see that writeSystem makes a system call for every character written, but writeLibc buffers characters and actually calls write() less frequently (for this program, only when it rewinds).

```
$ strace ./writeLibc
<lots of output>
$ strace ./writeSystem
<lots of output>
```

When you're done, turn in your source code for your completed writeSystem.c. Submit the file to the Gradescope assignment named EX01.