LOUISIANA STATE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CSC 4103: Operating Systems
Prof. Aisha Ali-Gombe
Programming Assignment # 2: Safe System Calls
Assign: February 11, 2025, Due: February 27, 2025 @ noon
Instruction – Students are required to work on their own or only with one PA buddies. Only one student can submit on behalf of all the group members

NO LATE SUBMISSIONS

As discussed in lecture, system calls provide a well-defined interface between applications running in user mode and the operating system kernel. Essentially, system calls "beg" the operating system to perform some operation, such as reading or writing a file, which must be tightly controlled for reasons of fairness, security, etc.

In this assignment, you'll improve the implementation of a fictitious set of system calls that provide file I/O. These system calls allow opening, closing, reading, and writing files. One interesting aspect of this set of system calls is that the position in a file for individual read and write operations is always specified (although the natural "current position" can be specified using the anchor CURRENT_POSITION…read on for more details). Another twist is that the system call for writing data into a file should prevent the string "\x7F""ELF" from appearing in the first four bytes of the file. That string is the binary character 0x7F followed by the three ASCII characters "ELF". The current implementation of the system calls appears in **`fileio.c`** and **`fileio.h`** (available from Moodle) and includes:

```
// open or create a file with pathname 'name' and return a File
// handle.  The file is always opened with read/write access. If the
// open operation fails, the global 'fserror' is set to OPEN_FAILED,
// otherwise to NONE.
File open_file(char *name);

// close a 'file'.  If the close operation fails, the global 'fserror'
// is set to CLOSE_FAILED, otherwise to NONE.
void close_file(File file);

// read at most 'num_bytes' bytes from 'file' into the buffer 'data',
// starting 'offset' bytes from the 'start' position.  The starting
// position is BEGINNING_OF_FILE, CURRENT_POSITION, or END_OF_FILE. If
// the read operation fails, the global 'fserror' is set to
// READ_FAILED, otherwise to NONE.
unsigned long read_file_from(File file, void *data, unsigned long
num_bytes, SeekAnchor start, long offset);
```

```
// write 'num_bytes' to 'file' from the buffer 'data', starting
// 'offset' bytes from the 'start' position.  The starting position
// is BEGINNING_OF_FILE, CURRENT_POSITION, or END_OF_FILE.  If an
// attempt is made to modify a file such that "\x7F""ELF" would appear
// in the first four bytes of the file, the write operation fails and
// ILLEGAL_ELF is stored in the global 'fserror'.  If the write fails
// for any other reason, the global 'fserror' is set to
// WRITE_FAILED, otherwise to NONE.
unsigned long write_file_at(File file, void *data, unsigned long
num_bytes, SeekAnchor start, long offset);
```

There's also a helper function that displays a string explanation of any error code that's raised by the system calls. That function is called **`fs_print_error()`.**

Your job is to improve the implementation of the special check that's performed in the **`write_file_at()`** system call to prevent "\x7F""ELF" from being stored in the first four bytes of a file. You should first study the implementations of the system calls, then carefully write some C programs to stress test both the current implementation and your modifications.

Some additional details:

- The current checks to prevent "\x7F""ELF" from being written to the file do not work in all cases!
- Don't change the names or parameters of the system calls. You must improve the implementation of "\x7F""ELF" checking without breaking the existing programming interface.
- Any write that wouldn't result in "\x7F""ELF" appearing in the first four bytes of the file is legal and **must** be allowed.
- Any write that would result in "\x7F""ELF" appearing in the first four bytes of the file is illegal and **must** result in an error, without modifying the file. You must **never** allow "\x7F""ELF" to appear in the first four bytes of the file, regardless of how clever a programmer using the system calls might be!
- The implementation must be efficient, e.g., **an implementation of the check for "\x7F""ELF" that reads the file on every write operation is not permitted**. Similarly, consuming large amounts of memory is also unacceptable.
- Your solution does **not** need to be thread-safe and you may assume that data may only be written into files using the API described in this document (e.g., no one will use an external editor, etc. to introduce data into any files).

Do your work and submit to the **classes.csc.lsu.edu** server via `ssh`.

When you create a test program, e.g., `readwrite.c`, compile and link your test program with the system call implementation with a command line like this:

```
$ gcc -o readwrite readwrite.c fileio.c
```

For example, one of my test programs, called `readwrite.c`, attempts a bunch of different file operations and interrogates the value of `fserror` after each operation via a call to `fs_print_error()`:

```
$ gcc -o readwrite readwrite.c fileio.c
$ ./readwrite
rm: important.dat: No such file or directory
Creating new file called "important.dat"...
FS ERROR: NONE
Writing something benign to beginning of file...
FS ERROR: NONE
Writing something evil to beginning of file...
FS ERROR: ILLEGAL_ELF: SHAME ON YOU!
Closing file...
FS ERROR: NONE
$
```

`readwrite.c` is available on Moodle and can be used as a starting point for developing your test programs.

SUBMISSION INSTRUCTIONS:

- **Students are required to work on their own or only with one PA buddies. Only one student can submit on behalf of all the group members.**
- When your group is ready to submit, organize your solution to this assignment in a directory called **`prog2`** in your 4103 accounts on **classes.csc.lsu.edu**. Please put the following files in that directory:
    - Your modified versions of `fileio.c` and `fileio.h`.
    - Any test programs you've developed.

Then issue the following command to turn in your solution:

```
$ ~ cs4103_ali/bin/p_copy 2
```

You MUST test your programs on the classes server, and you MUST submit electronically, by the due date!