Homework Assignment: Cross-Platform File Manipulation Library

Operating Systems Course

December 17, 2024

Introduction

The standard C library provides an abstraction for working with files through functions like `fopen()`, `fread()`, `fwrite()`, and `fclose()`. These functions make file manipulation simple and consistent across different operating systems by hiding platform-specific implementation details. For this assignment, you will take on the role of building a similar abstraction: a cross-platform file manipulation library, `so_file_lib`, which works seamlessly on both Linux and Windows.

The task is a valuable exercise in understanding how abstraction layers are implemented and how operating system APIs work under the hood. By the end of the assignment, you will have a deeper appreciation for how libraries like the C standard library provide such conveniences.

Task Description

Using the provided skeleton files, implement the following functions in `so_file_lib.c`:

1. `SO_FILE *so_fopen(const char *pathname, const char *mode);`
2. `int so_fclose(SO_FILE *stream);`
3. `size_t so_fread(void *ptr, size_t size, size_t count, SO_FILE *stream);`
4. `size_t so_fwrite(const void *ptr, size_t size, size_t count, SO_FILE *stream);`
5. `int so_fseek(SO_FILE *stream, long offset, int whence);`

6. `long so_ftell(SO_FILE *stream);`

You must ensure compatibility with both Linux and Windows systems. Use the `#ifdef` directive to include platform-specific headers and implement API calls accordingly.

Provided Documentation

For your reference, here is a summary of the key system calls and their platform-specific counterparts:

- **File Open:** `open()` (Linux) vs `CreateFile()` (Windows).
- **File Read:** `read()` (Linux) vs `ReadFile()` (Windows).
- **File Write:** `write()` (Linux) vs `WriteFile()` (Windows).
- **File Seek:** `lseek()` (Linux) vs `SetFilePointer()` (Windows).
- **File Close:** `close()` (Linux) vs `CloseHandle()` (Windows).

Each of these APIs has its nuances.

Development Tools Overview

You will need tools to compile and test your code on both platforms. Here is a brief description of the required tools:

- **GCC (GNU Compiler Collection):** A popular compiler on Linux for C/C++ programs, enabling the compilation of your code into an executable.
- **CMake:** A cross-platform build system generator. It helps ensure your project builds correctly on both Linux and Windows, regardless of the underlying compiler.
- **Microsoft Visual Studio:** The most widely used IDE on Windows, which includes the MSVC compiler (`cl.exe`).

Testing and Verification

To ensure correctness, a testing and compilation utility script, `helper.py`, has been provided. It includes the following features:

- **Compilation:** Use the command `python3 helper.py --compile` to compile your solution using CMake. This ensures the code builds correctly on both Linux and Windows.
- **Testing:** Use `python3 helper.py --check` to run a series of predefined tests that validate your implementation.

The tests will verify that your functions work as expected, including opening files, reading, writing, seeking, and closing them. Use the script to verify your solution before submission.

Submission Instructions

Submit an archive containing the completed `so_file_lib.c` file.

Evaluation Criteria

Your submission will be graded based on:

- Correctness: Implementation of the functions for both Linux and Windows.
- Testing: Evidence that your implementation works correctly on both platforms.
- Code Quality: Readability, proper use of comments, and adherence to coding standards.
- Code Safety: Make sure there are no memory leaks when the program exits. You can check for leaks using `valgrind`

References

- Linux API documentation: `https://man7.org/linux/man-pages/`
- Windows API documentation: `https://learn.microsoft.com/en-us/windows/win32/api/`

- GCC: `https://gcc.gnu.org/`
- Microsoft Visual Studio: `https://visualstudio.microsoft.com/`
- CMake: `https://cmake.org/`
- Valgrind: `https://valgrind.org/`