Chase Howard
2021-05-10
University of Washington
CSS 503B

Program 3: Stdio

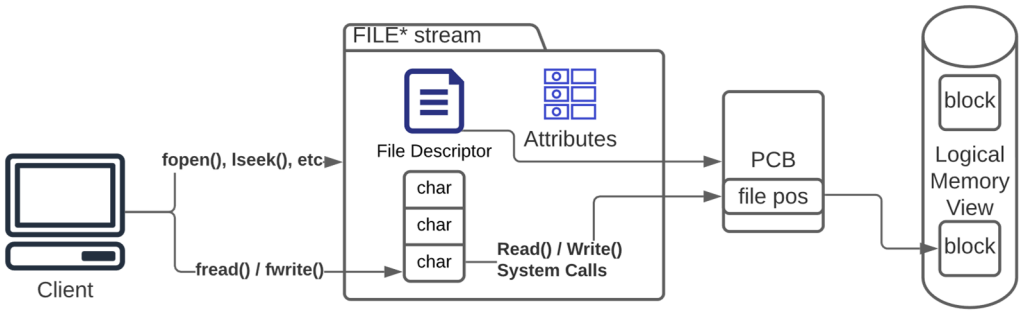
Purpose

The purpose of this assignment is to gain practice with improving performance by limiting costly system calls with the use of a single buffer for reading and writing to a file through a custom implementation of <stdio.h>. Additionally, gaining practice with file management.

Program Description

This programs functions add buffering and provide a user-friendly file stream interface (FILE *). The file stream interface is implemented by the standard I/O library utilizing the underlying system calls. When reading and/or writing small byte-counts (e.g., reading one line at a time from a file), buffered functions are faster by limiting the costly system calls.

The read() and write() system calls operate on file descriptors and read/write from/to buffers not strings. A file descriptor is just an integer referring to a currently open file. The OS uses that number as an index into the file descriptor table of files currently in use to access the actual device (e.g., disk, network, terminal).



On the other hand, file streams operators (fread/fwrite) interact directly with file streams: FILE *. File streams are dynamically allocated and allow reading/writing raw data. File stream operators, fread() and fwrite(), use the type void * since there are no data-specific requirements.

Discussion

Tests were run using a series of read and write methods to test the performance of Unix I/O, the submitted implementation of stdio.h and the original stdio.h. Test results from an average of 4 tests is listed below along with a short discussion to follow.

Table 1: Average run times comparing Unix and custom STDIO written for program 3 [Based on 5 runs ea.]

		Unix	My STDIO	Original STDIO
Read	Read Once	676	180	228
	Block Transfer	73	62	71
	Char Transfer	144159	1261	1798
	Random Transfer	146	96	94
Write	Write Once	84	111	91
	Block Transfer	114	115	136
	Char Transfer	157467	1319	1606
	Random Transfer	2575	166	176

General Performance observations:

- There was significant variation across tests – in order to have a statistically significant data set, this test would be run on a non-shared machine under a controlled environment over a large number of iterations. The data above was collected on a shared UW Linux machine and only includes data from 5 separate runs, in which the average is reported above.

Performance considerations against my own stdio.h and the original stdio.h

- Generally, run times from my implementation of stdio.h align with the performance of the original STDIO or are better. There are certainly deviations, specifically looking at a single write. There may be optimizations available there; however, I am unsure of what that may look like at the moment. There was a significant amount of variation from run to run, statistically these values may not hold any weight and considering the shared resource of a campus computer, the computation load can vary widely depending on when the process is run.
- With read commands, performance is improved generally across the entirety of test cases. The use of a buffer significantly drops the number of system calls required, which can see to make a significant impact at the Character test case
- With write commands there is a possible degradation in performance with my own stdio.h for writing once. This may be due to the additional overhead of buffer management on a single write call which eventually uses the same system call. This may be an opportunity for improvements / optimizations. Other test cases generally show agreement with or improvement upon the Unix I/O similar to the read operations.

Performance considerations against my stdio.h and the Unix-original stdio.h

- Performance comparisons against the Unix-original stdio.h are generally improved across the board with my own implementation. The use of buffers to limit system calls speeds up performance quite significantly.

Program Limitations and Improvements

Limitations

- When running ".\eval w r f test.txt", there will be an improper memory access during an printf call to delete a variable 'dec' – I don't believe this to be an issue of stdio but instead an issue with eval.cpp, possibly improper access to a file
- Flags set as '_IOLBF' and '_IONBF' flags currently ignored and treated as fully buffered. _IONBF should make direct system calls instead of populating buffer and _IOLBF should be buffered to newlines

Improvements

- Ideally, fgetc would call fread(), fputc() would call fwrite(), to improve modularity. Yet during testing I repeatedly was getting undesirable results.
- Additional methods provided in original stdio.h should be implemented
- Investigate techniques to optimize single write commands to more accurately match Unix I/O and the original stdio.h.
- Investigate method for tracking estimated runtime from a file – may help with some optimizations regarding the scheduler or buffer size.
- Investigate varying buffer size and how that affects performance, see if there is a method for tracking previous file operations to predict performance characteristics / optimize buffer size allocations.
- Proper header file, make fillBuffer() function private.