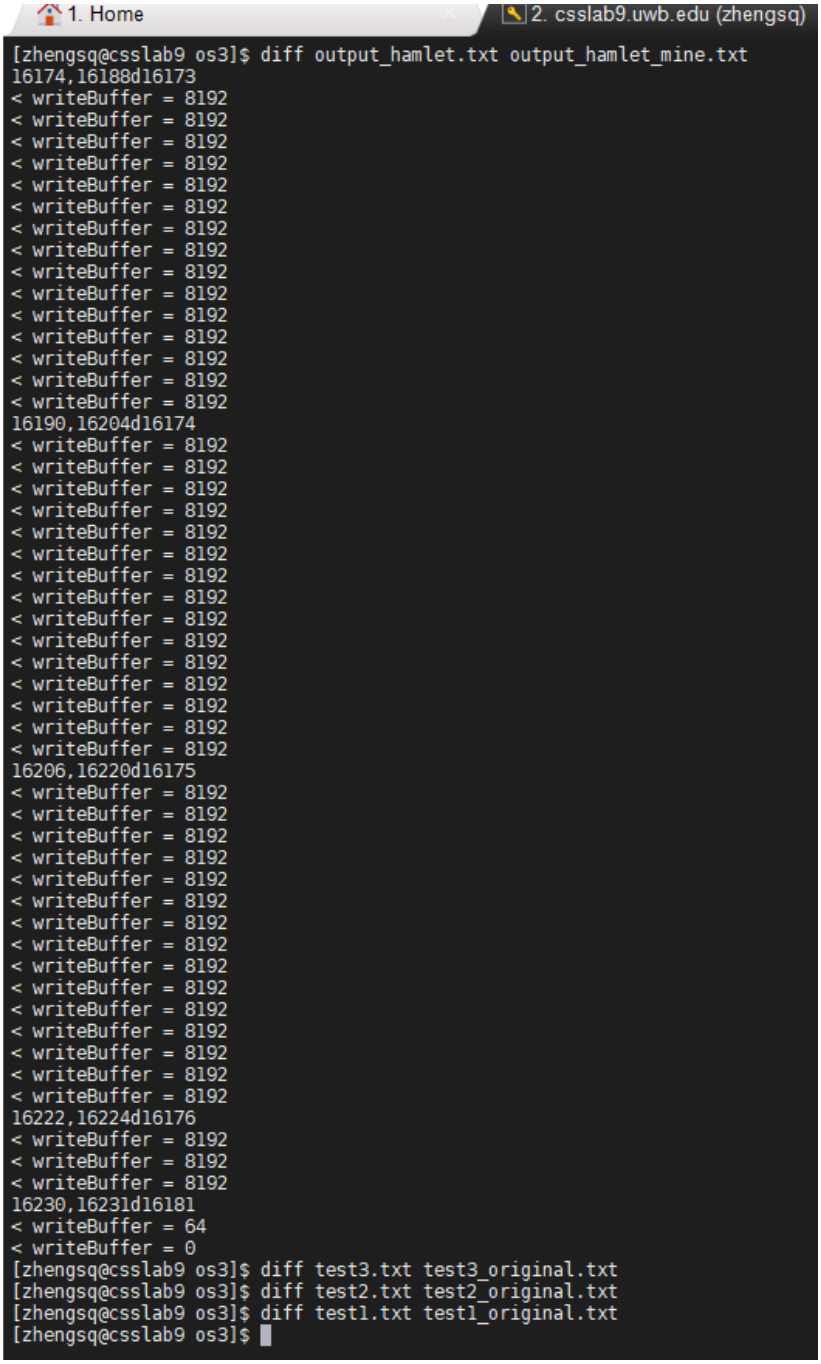
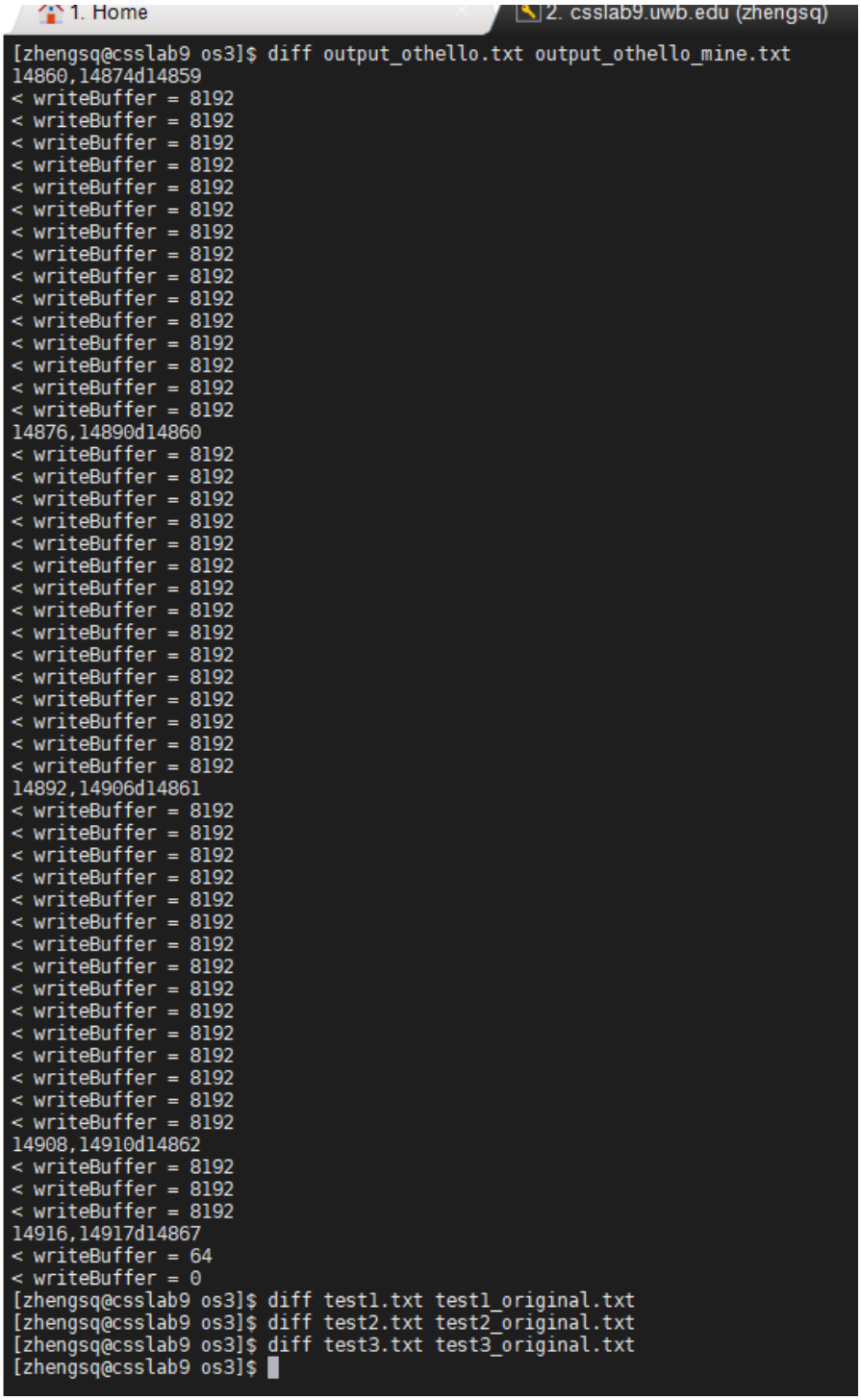
Step 4: comparing my output files vs. the originals : the only difference is my stdio file doesn't generate "writeBuffer = 8192" lines, which are not requirements, and we don't need concern about them.




1. Home
2. csslab9.uwb.edu (zhengsq)
[zhengsq@csslab9 os3]$ diff output_othello.txt output_othello_mine.txt
14860,14874d14859
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
14876,14890d14860
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
14892,14906d14861
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
14908,14910d14862
< writeBuffer = 8192
< writeBuffer = 8192
< writeBuffer = 8192
14916,14917d14867
< writeBuffer = 64
< writeBuffer = 0
[zhengsq@csslab9 os3]$ diff test1.txt test1_original.txt
[zhengsq@csslab9 os3]$ diff test2.txt test2_original.txt
[zhengsq@csslab9 os3]$ diff test3.txt test3_original.txt
[zhengsq@csslab9 os3]$

Step 5 : Test my implementation of stdio.h using the eval executable

```
[zhengsq@csslab9 os3]$ ./eval r u a hamlet.txt
Reads : Unix   I/O [Read   once     ] = 2008
[zhengsq@csslab9 os3]$ ./eval r u b hamlet.txt
Reads : Unix   I/O [Block  transfers] = 76
[zhengsq@csslab9 os3]$ ./eval r u c hamlet.txt
Reads : Unix   I/O [Char   transfers] = 138278
[zhengsq@csslab9 os3]$ ./eval r u r hamlet.txt
Reads : Unix   I/O [Random transfers] = 159
[zhengsq@csslab9 os3]$ ./eval r f a hamlet.txt
Reads : C File I/O [Read   once     ] = 171
[zhengsq@csslab9 os3]$ ./eval r f b hamlet.txt
Reads : C File I/O [Block  transfers] = 81
[zhengsq@csslab9 os3]$ ./eval r f c hamlet.txt
Reads : C File I/O [Char   transfers] = 16438
[zhengsq@csslab9 os3]$ ./eval r f r hamlet.txt
Reads : C File I/O [Random transfers] = 104
```

```
[zhengsq@csslab9 os3]$ ./eval w u a test.txt
Writes: Unix   I/O [Read   once     ] = 109
[zhengsq@csslab9 os3]$ ./eval w u b test.txt
Writes: Unix   I/O [Block  transfers] = 127
[zhengsq@csslab9 os3]$ ./eval w u c test.txt
Writes: Unix   I/O [Char   transfers] = 163258
[zhengsq@csslab9 os3]$ ./eval w u r test.txt
Writes: Unix   I/O [Random transfers] = 2563
[zhengsq@csslab9 os3]$ ./eval w f a test.txt
Writes: C File I/O [Read   once     ] = 104
[zhengsq@csslab9 os3]$ ./eval w f b test.txt
Writes: C File I/O [Block  transfers] = 131
[zhengsq@csslab9 os3]$ ./eval w f c test.txt
Writes: C File I/O [Char   transfers] = 2145
[zhengsq@csslab9 os3]$ ./eval w f r test.txt
Writes: C File I/O [Random transfers] = 211
[zhengsq@csslab9 os3]$
```

Step 6: replace the first line of the eval.cpp, (i.e., “stdio.h”) with <stdio.h>

```
[zhengsq@csslab9 os3]$ cd libVer
[zhengsq@csslab9 libVer]$ ./compile.sh
[zhengsq@csslab9 libVer]$ ./eval r f a hamlet.txt
Reads : C File I/O [Read   once     ] = 1964
[zhengsq@csslab9 libVer]$ ./eval r f b hamlet.txt
Reads : C File I/O [Block  transfers] = 2993
[zhengsq@csslab9 libVer]$ ./eval r f c hamlet.txt
Reads : C File I/O [Char   transfers] = 1396
[zhengsq@csslab9 libVer]$ ./eval r f r hamlet.txt
Reads : C File I/O [Random transfers] = 108
[zhengsq@csslab9 libVer]$ ./eval w f a test.txt
Writes: C File I/O [Read   once     ] = 102
[zhengsq@csslab9 libVer]$ ./eval w f b test.txt
Writes: C File I/O [Block  transfers] = 129
[zhengsq@csslab9 libVer]$ ./eval w f c test.txt
Writes: C File I/O [Char   transfers] = 1217
[zhengsq@csslab9 libVer]$ ./eval w f r test.txt
Writes: C File I/O [Random transfers] = 153
[zhengsq@csslab9 libVer]$
```

Documentation of stdio.cpp implementation:

The goal of my **stdio** class is to emulate the standard I/O library stdio, which is an architecture independent library that allows C/C++ programs to read and write files instead of directly calling the underlying OS system calls.

The implemented functions add buffering and provide a user-friendly file stream interface (FILE *). The file stream interface is implemented by 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.

Upon a file open, fopen() returns a pointer to a FILE object that maintains the attributes of the opened file.

The 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.

The function fflush() is to synchronize an output stream with the actual file, which means transfer data in buffer into disk. The setbuf() and setvbuf() is to set the size of an input / output stream buffer either by the user or the system, depending on different modes and situations. The fpurge() is used to clear an input / output stream buffer. The fgetc()/fgets() and fputc()/fputs() read/write a character(string) from/to a file stream, they are actually the specific implements of fread() and fwrite(). The fseek() moves the file position to a specific location in a file in the disk.

Besides the functions implemented above, I also emulate various file-opening modes, according to respective modes within system calls, thus making my stdio.h close to real Unix system calls.

r or rb = O_RDONLY
w or wb = O_WRONLY | O_CREAT | O_TRUNC
a or ab = O_WRONLY | O_CREAT | O_APPEND
r+ or rb+ or r+b = O_RDWR
w+ or wb+ or w+b = O_RDWR | O_CREAT | O_TRUNC
a+ or ab+ or a+b = O_RDWR | O_CREAT | O_APPEND

Then, I tested my implementation of stdio.h using the driver executable, eval executable and the texts provided on canvas: hamlet.txt, othello.txt, test1.txt, test2.txt, test3.txt, to verify correctness.

Limitation and possible extension of your program

This implementation doesn't consider too much about error handling issues, so there still exists some slight difference with Unix-original library upon error happening, my version cannot output specific error codes in line with how <stdio.h> does.

Another limitation is the implementation cannot be used upon multi-programming. I didn't consider multi-threads synchronization issues, e.g., locks and conditions weren't utilized as means to solve potential problems where multi-threads possibly read and write into buffer and disk at the same time.

Performance consideration between your own stdio.h and Unix I/O

In addition to utilizing Unix I/O system calls, my own stdio.h adds buffering. When reading and/or writing small byte-counts (e.g., reading one line at a time from a file), buffered functions are faster.
Unix I/O system call, e.g. write() is implemented by the interface between user mode and the operating system kernel, but context switch is very expensive, based on this bottleneck, my implementation fwrite() buffer the output, by default until end-of-line is reached. When the buffer is full or terminated with a newline, it is written to the file via a call fflush() to write from the buffer.

Performance consideration between your own stdio.h and the Unix-original stdio.h

Through comparing executing ./eval command using my own stdio.h and the Unix-original stdio.h, I got results shown on the screenshots.

command	description	#include "stdio.h"	#include <stdio.h>
./eval r f a hamlet.txt	read hamlet.txt at once.	171	1964
./eval r f b hamlet.txt	read hamlet.txt every 4096 bytes	81	2993
./eval r f c hamlet.txt	read hamlet.txt one by one character	16438	1396
./eval r f r hamlet.txt	read hamlet.txt with random sizes	104	108
./eval w f a test.txt	write to test.txt at once	104	102
./eval w f b test.txt	write to test.txt every 4096 bytes	131	129
./eval w f c test.txt	write to test.txt one by one character	2145	1217
./eval w f r test.txt	write to test.txt with random sizes	211	153

We can see, for reading a big chunk of file at one time, my implementation wins, but for reading one by one character, my implementation seems taking much more time. For writing functions, my implementation and Unix-original are roughly the same.