

14 WANT ANSWERS



Latest activity: 15 May

## QUESTION TOPICS

C (programming language)

Computer Programming

Edit Topics

## SHARE QUESTION

Twitter

Facebook

## QUESTION STATS

Views 23,070

Want Answers 14

Edits

## Why do we use the functions fflush(stdin) and fflush(stdout) in c?

Want Answers 14

Comment Share Downvote

...

5 ANSWERS

ASK TO ANSWER

**Jian Sun**

Edit Biography • Make Anonymous

Write your answer, or answer later

**Adrien Lucas Ecoffet**, C was my first programming language

45 upvotes by Anders Kaseorg, John Matthew, Neel Hajare, (more)

You would use `fflush(stdout)` to ensure that whatever you just wrote in a file/the console is indeed written out on disk/the console.

The reason is that actually writing, whether to disk, to the terminal, or pretty much anywhere else, is pretty slow. Further, writing 1 byte takes roughly the same time as writing, say, a few hundred bytes[1]. Because of this, data you write to a stream is actually stored in a buffer which is flushed when it is full or when you call `fflush`. Calling `fflush` means you are accepting that your function call will take a bit of time but that you are 100% sure that you want this out right away.

As for `fflush(stdin)`, the reason people call it is that some functions, `scanf` in particular, sometimes leave the input in a counter-intuitive state.

For instance, say you call `scanf("%d", &n)` and then `fgets(buf, 100, stdin)`, expecting an input of the form:

```
1 10
2 line
```

Well it turns out that what you will get is an empty line on the `fgets` because `scanf` didn't move you past the `\n` after it read the 10.

Because of that kind of problem, people use `fflush(stdin)`. This is because at the time you call it, the `\n` following "10" is in the buffer but not the string "line\n". **THIS IS A TERRIBLE IDEA. IT ONLY WORKS ON WINDOWS.**

In it is explicitly said in the standard that `fflush` should only apply to output streams. This is because things that just don't make sense would happen if you used it out input streams.

See, there is absolutely nothing that indicates that "line\n" will not be in the buffer at that time. Sure it works when writing input manually, but this might fail if the user is copying and pasting into your program, and is even less likely to work if someone redirects a file as input to your program. Heck, even just a fast typer, a slow computer and a bit of luck might mess this up...

Because of this it seems, the implementation on my Mac simply ignores `fflush(stdin)`. I don't know what happens on other platforms but I'm sure there's at least one that has the behavior I described earlier where you might

## RELATED QUESTIONS

C (programming language)what is the use of fflush() in c?

Is fflush(stdin) unsafe to use?

What should I use instead of using fflush(stdin) in C?

What should I use instead of fflush(stdin) when scanf or similar functions refuse to read past a \n?

What is the fastest way to get input from STDIN / give output to STDOUT in C/C++?

How are stdin and stdout implemented?

Why is use of pointers necessary in C for swapping of two numbers through a function call?

Is there any function in Java like fflush(stdin) in C++?

What is fflush(stdin); in a C language?

I never realized fflush(stdin) was wrong until it failed on me. What other commonly used things in C and C++ will usually...

More Related Questions

just lose some of your input because of an `fflush`. Point is, don't use it on input streams.

[1] The exact number would depend on the machine and the type of output.

Written 31 Jul, 2013. 21,268 views.

Upvote | 45

Downvote Comment 1 Share

...



**Vaisakh Sudheesh**, Geek :) working on embedded systems

5 upvotes by Siddharth Goel, Jason Jee, Rohit Pratap Singh, (more)

If i remember correctly, most of the input-output routines are buffered, i.e. they may printout or accept the IO values only on encountering a new line or reaching the buffer limit.

So in-order to flush the data stored in IO buffer, we need to use **`fflush`** function.

Written 31 Jul, 2013. 5,671 views.

Upvote | 5

Downvote Comment 1 Share

...



**Arindam Mukherjee**

4 upvotes by Shubham Shrinath, Shivali Vij, Nitish Chandra, (more)

Let us first understand the different I/O functions that the standard library provides and their relationship to each other.

### Output

For formatted output, you have *`fprintf`* / *`printf`* / and their variants.

For string output, you have *`fputs`*.

For output of uninterpreted data i.e. raw bytes, you have *`fwrite`*.

These are all library functions that are most likely written in the C language and call into some operating system service for doing the actual I/O.

### System calls

On Unix, the operating system service that the output functions call into would be the *`write`* system call. A system call is available to user processes in the form of a wrapper provided by the platform's C library (e.g. `glibc` on Linux). That wrapper marshals the passed arguments into registers and initiates a software interrupt to execute the actual system call, which is code in the kernel. That system call picks up the arguments from the registers and does stuff which could involve interacting with some device (like a disk). For all of this boilerplate plumbing and switch to kernel mode, system calls have an overhead.

### `fwrite` -> `write`

Let's focus on the `fwrite` library function and `write` system call. On my Linux system, their prototypes look like this:

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

```
ssize_t write(int fd, const void *buf, size_t count);
```

So when you want to write a chunk of data in a byte array identified by `ptr`, you call `fwrite` with `ptr` pointing to your data buffer, `size` set to the size of each logical element in the array, `nmemb` set to the number of such elements to write to and stream pointing (hopefully) to an open stream to some device (disk file, terminal, etc.).

This should really translate to a call to `write`. However there isn't a one-to-one correspondence between each `fwrite` and `write` call. What the `write` call really

does is to copy count bytes from a user buffer pointed to by `buf` from user space to a buffer in a cache of buffers in the kernel associated with the (hopefully) open file descriptor `fd` - which is some positive integer pointing to an open file. No actual write happens.

### Buffering

The way `fwrite` handles this translation is to maintain an internal buffer in the library and when it fills up, pass the whole buffer to write. System calls are costly to make and this ensures that you aren't making too many of them notwithstanding how randomly sized your buffers were that you passed to `fwrite`.

You could override this behaviour by calling `fflush` on an output stream. Then it will call `write` on the current state of its buffer without waiting for it to fill up.

You could even override the library's default buffer and provide your own buffer with `setbuf` or `setvbuf` library functions. You could also turn off buffering - it would slow you down if you did that for disk I/O though.

### Writing to the disk

The actual writing to the disk happens when the kernel's I/O scheduler identifies dirty buffers that need to be flushed and does some optimization to minimize disk I/O overhead and writes them out to the disk. In Linux, there is a dedicated thread called `pdflush` which does this, I think every 30 seconds. You could still force the issue by calling `fsync` or `fdatasync` system calls.

### fflush details

Now `fflush` is just a C library function (like `fwrite`, `setvbuf`, etc.) which does an out of turn transfer of the user's buffer to the kernel by calling `write` even when the user's buffer hadn't filled up. It's an override.

So when you do `fflush(stdout)` or `fflush(any_output_stream)`, if that stream be buffered, its user buffer would be passed to the kernel via a `write` system call. It wouldn't immediately be written to the disk.

`fflush(stdin)` or `fflush(any_input_stream)` is spelled out as undefined by the C Standard. So we shouldn't use it, its non-portable and do not rely on it.

If you want to discard unread input, read the necessary number of characters and discard them.

Written 5 Mar, 2014. 4,032 views.

Upvote | 4

Downvote Comment Share

...



### Sameer Gupta

5 upvotes by Lillian Xiong, Anjali Pardeshi, Kiran Kannar, (more)

To flush the fork buffers, you use `fflush()` or `\n` or `read`, so as you don't get the same output twice. e.g.

```
1 /* include <the needful> */
2 int main()
3 {
4     printf ("foo");
5     fork();
6 }
```

will print foofoo, till you use `\n` or `fflush` or issue.

There can be other instances, where a buffer is generated in the process memory and flushing it is important for some purpose.

Point remains, that you or unix use `fflush` to flush the buffers.

Written 31 Oct, 2014. 4,081 views.

Upvote | 5

Downvote Comment Share

...



Shri Dsouza, Computer Programming

The C library function **int fflush(FILE \*stream)** flushes the output buffer of a stream.

Declaration

Following is the declaration for fflush() function.

```
int fflush(FILE *stream)
```

#### Parameters

- **stream** – This is the pointer to a FILE object that specifies a buffered stream.

#### Return Value

This function returns a zero value on success. If an error occurs, EOF is returned and the error indicator is set (i.e. feof).

#### Example

The following example shows the usage of fflush() function.

```
#include <stdio.h>
#include <string.h>
int main()
{
    char buff[1024];
    memset( buff, '\0', sizeof( buff ));
    fprintf(stdout, "Going to set full buffering on\n");
    setvbuf(stdout, buff, _IOFBF, 1024);
    fprintf(stdout, "This is the program\n");
    fprintf(stdout, "This output will go into buff\n");
    fflush( stdout );
```

```
sleep(5);
return(0);
}
```

Let us compile and run the above program that will produce the following result. Here program keeps buffering into the output into **buff** until it faces first call to **fflush()**, after which it again starts buffering the output and finally sleeps for 5 seconds. It sends remaining output to the STDOUT before program comes out.

Going to set full buffering on This output will go into buff and this will appear as when program will come after sleeping 5 seconds.  
hope so answer is helpful.

Written 15 May. 139 views. Asked to answer by Srinivas Pavan Kumar.

[Upvote](#)
[Downvote](#) [Comment](#) [Share](#)

...

[Home](#)
[Write](#)
[Notifications](#)

5


[Jian](#)
[Add Question](#)

## Top Stories from Your Feed

Answer written • Mon

### Working Hard: How does it feel to buy a Lamborghini (or any dream car) from your own salary?



Tom Cruz, Founder of AptoHQ.com  
3.4k upvotes by Michael Cohen, Georgy Kishtoo, Zonaib Siddiqui, (more)

Answer written • 19 May

### For airline pilots traveling as passengers, what things do you pay attention to during a flight that most passengers might not?



John Chesire, 40 Years of aviation experience, incl...

968 upvotes by Mike Holovacs (Former 2W0X1 (Munitions Systems) crewmember and... ), Tom Farrier (Retired US Air Force command pilot; Current avi... ), Nelson Brown, (more)  
For one thing, I always look around to find the nearest emergency exit. Then I count the number

Answer written • Mon


### Startups: What is the best advice for a young, first-time startup CEO?



James Altucher, Blogger, author, social media, invest... ✓

1.9k upvotes by Rupert Baines (Founded several start-ups, exec at several more... ), Bernie Klinder, Aashutosh Shah, (more)

This is going to be a bullet FAQ on starting a business. I've started about 20 businesses. Some major successes. Some major failures. I'm also invested in about 30 startups. I've seen some blow



Read In Feed

of seats between me and that exit. It only takes a

Read In Feed

u...

Read In Feed