Understanding the need of fflush() and problems associated with it

Below is a sample code for using fflush().

```
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <io.h>

void flush(FILE *stream);

int main(void)
{
   FILE *stream;
   char msg[] = "This is a test";

   /* create a file */
   stream = fopen("DUMMY.FIL", "w");

   /* write some data to the file */
   fwrite(msg, strlen(msg), 1, stream);

   clrscr();
   printf("Press any key to flush DUMMY.FIL:");
   getch();

   /* flush the data to DUMMY.FIL without closing it */
   flush(stream);

   printf("\nFile was flushed, Press any key to quit:");
   getch();
   return 0;
}

void flush(FILE *stream)
{
     int duphandle;

     /* flush the stream's internal buffer */
     fflush(stream);

     /* make a duplicate file handle */
     duphandle = dup(fileno(stream));

     /* close the duplicate handle to flush the DOS buffer */
     close(duphandle);
}
```

All I know about the fflush() is that it is a library function used to do flush output buffer. I want to know that what is basic purpose of using fflush(), where can I use it. And mainly I am interested in knowing that `what can be problems with using fflush()`.

c fflush

edited Feb 25 at 12:35

Tshepang
3,542 ● 9 ● 46 ● 91

asked May 27 '13 at 21:45

Karan Mer
1,317 ● 7 ● 30

when you use `printf()` without any new line, it will probably won't print immediately. if you know your program may crash any second, you can use `fflush()` (or newlines...). I don't think this is the common usage though. – Elazar May 27 '13 at 21:48

Only problem I can think of is that doing it too often is bad for performance. Also, you should know that closing a file or program termination implies an automatic flush. A short-running program will seldom need to call `fflush()` explicitly. – thejh May 27 '13 at 22:04

3 Answers

It's a little hard to say what "can be problems with" (excessive?) use of `fflush`. All kinds of things *can* be, or become, problems, depending on your goals and approaches. Probably a better way to look at this is what the intent of `fflush` is.

The first thing to consider is that `fflush` is defined only on output streams. An output stream collects "things to write to a file" into a large(ish) buffer, and then writes that buffer to the file. The point of this collecting-up-and-writing-later is to improve speed/efficiency, in two ways:

- On modern OSes, there's some penalty for crossing the user/kernel protection boundary (the system has to change some protection information in the CPU, etc). If you make a large number of OS-level write calls, you pay that penalty for each one. If you collect up, say, 8192 or so individual writes into one large buffer and then make one call, you remove most of that overhead.
- On many modern OSes, each OS write call will try to optimize file performance in some way, e.g., by discovering that you've extended a short file to a longer one, and it would be good to move the disk block from point A on the disk to point B on the disk, so that the longer data can fit contiguously. (On older OSes, this is a separate "defragmentation" step you might run manually. You can think of this as the modern OS doing dynamic, instantaneous defragmentation.) If you were to write, say, 500 bytes, and then another 200, and then 700, and so on, it will do a lot of this work; but if you make one big call with, say, 8192 bytes, the OS can allocate a large block once, and put everything there and not have to re-defragment later.

So, the folks who provide your C library and its stdio stream implementation do whatever is appropriate on your OS to find a "reasonably optimal" block size, and to collect up all output into chunk of that size. (The numbers 4096, 8192, 16384, and 65536 often, today, tend to be good ones, but it really depends on the OS, and sometimes the underlying file system as well. Note that "bigger" is not always "better": streaming data in chunks of four gigabytes at a time will probably perform worse than doing it in chunks of 64 Kbytes, for instance.)

But this creates a problem. Suppose you're writing to a file, such as a log file with date-and-time stamps and messages, and your code is going to keep writing to that file later, but right now, it wants to suspend for a while and let a log-analyzer read the current contents of the log file. One option is to use `fclose` to close the log file, then `fopen` to open it again in order to append more data later. It's more efficient, though, to push any pending log messages to the underlying OS file, but keep the file open. That's what `fflush` does.

Buffering also creates another problem. Suppose your code has some bug, and it sometimes crashes but you're not sure if it's about to crash. And suppose you've written something and it's very important that *this* data get out to the underlying file system. You can call `fflush` to push the data through to the OS, before calling your potentially-bad code that might crash. (Sometimes this is good for debugging.)

Or, suppose you're on a Unix-like system, and have a `fork` system call. This call duplicates the entire user-space (makes a clone of the original process). The stdio buffers are in user space, so the clone has the same buffered-up-but-not-yet-written data that the original process had, at the time of the `fork` call. Here again, one way to solve the problem is to use `fflush` to push buffered data out just before doing the `fork`. If everything is out before the `fork`, there's nothing to duplicate; the fresh clone won't ever attempt to write the buffered-up data, as it no longer exists.

The more `fflush`-es you add, the more you're defeating the original idea of collecting up large chunks of data. That is, you are making a tradeoff: large chunks are more efficient, but are causing some other problem, so you make the decision: "be less efficient here, to solve a problem more important than mere efficiency". You call `fflush`.

Sometimes the problem is simply "debug the software". In that case, instead of repeatedly calling `fflush`, you can use functions like `setbuf` and `setvbuf` to alter the buffering behavior of a stdio stream. This is more convenient (fewer, or even no, code changes required—you can control the set-buffering call with a flag) than adding a lot of `fflush` calls, so that could be considered a "problem with use (or excessive-use) of `fflush`".

answered May 27 '13 at 23:31

torek
39.9k ● 2 ● 42 ● 67

Well, @torek's answer is almost perfect, but there's one point which is not so accurate.

> The first thing to consider is that fflush is defined only on output streams.

According to man fflush, fflush can also be used in **input** streams:

> For output streams, fflush() forces a write of all user-space buffered data for the given output

> or update stream via the stream's underlying write function. **For input streams, fflush() discards any buffered data that has been fetched from the underlying file, but has not been consumed by the application**. The open status of the stream is unaffected. So, when used in input, fflush just discard it.

Here is a demo to illustrate it:

```
#include<stdio.h>

#define MAXLINE 1024

int main(void) {
  char buf[MAXLINE];

  printf("prompt: ");
  while (fgets(buf, MAXLINE, stdin) != NULL)
    fflush(stdin);
    if (fputs(buf, stdout) == EOF)
      printf("output err");

  exit(0);
}
```

edited Mar 2 at 15:41

answered Mar 2 at 15:17

jaseywang
11 • 3

`fflush()` empties the buffers related to the stream. if you e.g. let a user input some data in a very shot timespan (milliseconds) and write some stuff into a file, the writing and reading buffers may have some "reststuff" remaining in themselves. you call `fflush()` then to empty all the buffers and force standard outputs to be sure the next input you get is what the user pressed then.

reference: http://www.cplusplus.com/reference/cstdio/fflush/

answered May 27 '13 at 21:59

TheOneAndOnly
200 • 9

what can be the problems in using it? – Karan Mer May 27 '13 at 22:02

`fflush()` is a standard C function and *is* supported on Linux. – Blue Moon May 27 '13 at 22:17

This answer is completely wrong and suggests that `fflush` can be used for input. It cannot. – R.. May 28 '13 at 2:41

It does not suggest that Oo it says that it empties the buffers, that are connected with the streams *to make sure your next input becomes right* – TheOneAndOnly May 28 '13 at 14:28