Difference between fflush and fsync

I thought fsync() does fflush() internally so using fsync() on a stream is OK. But i am getting unexpected result when executed under network I/O.

My code snippet:

```
FILE* fp = fopen(file,"wb");
 /* multiple fputs() call like: */
 fputs(buf, fp);
 ...
 ...
 fputs(buf.c_str(), fp);
 /* get fd of the FILE pointer */
 fd = fileno(fp);
 #ifndef WIN32
 ret = fsync(fd);
 #else
 ret = _commit(fd);
 fclose(fp);
```

But it seems _commit() is not flushing the data (i tried on Windows and the data was written on Linux exported filesystem).

When i changed the code as:

```
FILE* fp = fopen(file,"wb");
  /* multiple fputs() call like: */
  fputs(buf, fp);
  ...
  ...
  fputs(buf.c_str(), fp);
  /* fflush the data */
  fflush(fp);
  fclose(fp);
```

This time it flushes the data.

I am wondering if _commit() does the same thing as fflush(). Any inputs?

c windows fsync fflush

edited Feb 26 '10 at 9:42

nos
117k ● 27 ● 188 ● 317

asked Feb 26 '10 at 9:36

Adil
715 ● 2 ● 15 ● 27

What is the problem you were seeing with the first example? – rogerdpack Nov 8 '12 at 0:16

1 @rogerdpack in first example, writing to stream through fputs() is not synching / committing to disk even if calling _commit() function on the fd (file descriptor). This test was done under cluster system where the remote linux filesystem is exported as CIFS and used on Windows machine and node failover is tested during write. When node recovers it has been found that file size is zero. – Adil Nov 9 '12 at 12:26

3 Answers

`fflush()` works on FILE* , it just flushes the internal buffers in the `FILE*` of your application out to the OS.

`fsync` works on a lower level, it tells the OS to flush its buffers to the physical media.

OSs heavily caches data you write to a file. If the OS enforced every write to hit the drive, things would be *very* slow. fsync(among other things) allows you to control when the data should hit the drive.

Furthermore, fsync/commit works on a file descriptor. It has no knowledge of a FILE* and can't flush its buffers. FILE* lives in your application, file descriptors lives in the OS kernel, typically.

answered Feb 26 '10 at 9:41


nos
117k • 27 • 188 • 317

Thanks i was thinking on the same line. So if we are using FILE* then the same can be achieved by fflush() followed by fsync(). – Adil Feb 26 '10 at 9:48

No, because you cannot `fsync` a `FILE*`. – patrickvacek Mar 26 '14 at 16:37

1 @patrickvacek actually you can get the file descriptor from `FILE *` using `int fileno(FILE * stream);` from `stdio.h`. – jotik Aug 11 '14 at 13:48

@jotik, fair point. That should work. – patrickvacek Aug 11 '14 at 16:02

@jotik: You should use *either* (standard) `FILE *` functions, *or* (operating system) file handles. You don't mix them. And `fileno()` is not a standard function. Unfortunately, people have been notoriously lax with "expanding" standard headers... – DevSolar Feb 4 at 15:32

I could say that for simplicity:

use `fsync()` with not streaming files (integer file descriptors)

use `fflush()` with file streams.

Also here is the help from man:

```
int fflush(FILE *stream); // flush a stream, FILE* type

int fsync(int fd); // synchronize a file's in-core state with storage device
                   // int type
```

answered Feb 4 at 15:29


pulse
28 • 6

To force the commitment of recent changes to disk, use the sync() or fsync() functions.

fsync() will synchronize all of the given file's data and metadata with the permanent storage device. It should be called just before the corresponding file has been closed.

sync() will commit all modified files to disk.

answered Mar 5 at 14:24


user3428045
1