## NAME

sync_file_range – sync a file segment with disk

## SYNOPSIS

```
#define _GNU_SOURCE         /* See feature_test_macros(7) */
#include <fcntl.h>

int sync_file_range(int fd, off64_t offset, off64_t nbytes,
                    unsigned int flags);
```

## DESCRIPTION

**sync_file_range**() permits fine control when synchronizing the open file referred to by the file descriptor *fd* with disk.

*offset* is the starting byte of the file range to be synchronized. *nbytes* specifies the length of the range to be synchronized, in bytes; if *nbytes* is zero, then all bytes from *offset* through to the end of file are synchronized. Synchronization is in units of the system page size: *offset* is rounded down to a page boundary; *(offset+nbytes-1)* is rounded up to a page boundary.

The *flags* bit-mask argument can include any of the following values:

**SYNC_FILE_RANGE_WAIT_BEFORE**
: Wait upon write-out of all pages in the specified range that have already been submitted to the device driver for write-out before performing any write.

**SYNC_FILE_RANGE_WRITE**
: Initiate write-out of all dirty pages in the specified range which are not presently submitted write-out. Note that even this may block if you attempt to write more than request queue size.

**SYNC_FILE_RANGE_WAIT_AFTER**
: Wait upon write-out of all pages in the range after performing any write.

Specifying *flags* as 0 is permitted, as a no-op.

### Warning

This system call is extremely dangerous and should not be used in portable programs. None of these operations writes out the file's metadata. Therefore, unless the application is strictly performing overwrites of already-instantiated disk blocks, there are no guarantees that the data will be available after a crash. There is no user interface to know if a write is purely an overwrite. On filesystems using copy-on-write semantics (e.g., *btrfs*) an overwrite of existing allocated blocks is impossible. When writing into preallocated space, many filesystems also require calls into the block allocator, which this system call does not sync out to disk. This system call does not flush disk write caches and thus does not provide any data integrity on systems with volatile disk write caches.

### Some details

**SYNC_FILE_RANGE_WAIT_BEFORE** and **SYNC_FILE_RANGE_WAIT_AFTER** will detect any I/O errors or **ENOSPC** conditions and will return these to the caller.

Useful combinations of the *flags* bits are:

**SYNC_FILE_RANGE_WAIT_BEFORE | SYNC_FILE_RANGE_WRITE**
: Ensures that all pages in the specified range which were dirty when **sync_file_range**() was called are placed under write-out. This is a start-write-for-data-integrity operation.

**SYNC_FILE_RANGE_WRITE**
: Start write-out of all dirty pages in the specified range which are not presently under write-out. This is an asynchronous flush-to-disk operation. This is not suitable for data integrity operations.

**SYNC_FILE_RANGE_WAIT_BEFORE** (or **SYNC_FILE_RANGE_WAIT_AFTER**)
: Wait for completion of write-out of all pages in the specified range. This can be used after an earlier **SYNC_FILE_RANGE_WAIT_BEFORE | SYNC_FILE_RANGE_WRITE** operation to wait for completion of that operation, and obtain its result.

**SYNC_FILE_RANGE_WAIT_BEFORE | SYNC_FILE_RANGE_WRITE | SYNC_FILE_RANGE_WAIT_AFTER**

This is a write-for-data-integrity operation that will ensure that all pages in the specified range which were dirty when **sync_file_range**() was called are committed to disk.

## RETURN VALUE

On success, **sync_file_range**() returns 0; on failure −1 is returned and *errno* is set to indicate the error.

## ERRORS

**EBADF**

*fd* is not a valid file descriptor.

**EINVAL**

*flags* specifies an invalid bit; or *offset* or *nbytes* is invalid.

**EIO** I/O error.

**ENOMEM**

Out of memory.

**ENOSPC**

Out of disk space.

**ESPIPE**

*fd* refers to something other than a regular file, a block device, or a directory.

## VERSIONS

**sync_file_range**() appeared on Linux in kernel 2.6.17.

## CONFORMING TO

This system call is Linux-specific, and should be avoided in portable programs.

## NOTES

**sync_file_range2()**

Some architectures (e.g., PowerPC, ARM) need 64-bit arguments to be aligned in a suitable pair of registers. On such architectures, the call signature of **sync_file_range**() shown in the SYNOPSIS would force a register to be wasted as padding between the *fd* and *offset* arguments. (See **syscall**(2) for details.) Therefore, these architectures define a different system call that orders the arguments suitably:

```
int sync_file_range2(int fd, unsigned int flags,
                     off64_t offset, off64_t nbytes);
```

The behavior of this system call is otherwise exactly the same as **sync_file_range**().

A system call with this signature first appeared on the ARM architecture in Linux 2.6.20, with the name **arm_sync_file_range**(). It was renamed in Linux 2.6.22, when the analogous system call was added for PowerPC. On architectures where glibc support is provided, glibc transparently wraps **sync_file_range2**() under the name **sync_file_range**().

## SEE ALSO

**fdatasync**(2), **fsync**(2), **msync**(2), **sync**(2)

## COLOPHON

This page is part of release 5.02 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man−pages/.