Async/sync/fsync/fdatasync

2018年5月11日　　14:13

使用区别？
实现差异？

File open flags:

O_SYNC
Write operations on the file will complete according to the requirements of synchronized I/O *file* integrity completion (by contrast with the synchronized I/O *data* integrity completion provided by **O_DSYNC**.)
By the time write(2) (or similar) returns, the output data and associated file metadata have been transferred to the underlying hardware (i.e., as though each write(2) was followed by a call to fsync(2)).

O_DSYNC
Write operations on the file will complete according to the requirements of synchronized I/O *data* integrity completion.
By the time write(2) (and similar) return, the output data has been transferred to the underlying hardware, along with any file metadata that would be required to retrieve that data(i.e., as though each write(2) was followed by a call to fdatasync(2)).

O_ASYNC
Enable signal-driven I/O: generate a signal (**SIGIO** by default,
but this can be changed via fcntl(2)) when input or output becomes possible on this file descriptor. This feature is available only for terminals, pseudoterminals, sockets, and (since Linux 2.6) pipes and FIFOs. See fcntl(2) for further details.

系统调用：
Sync：唤醒disk flush线程，将dirty数据提交到io队列，总是返回成功

```
SYSCALL_DEFINE0(sync)
{
	int nowait = 0, wait = 1;

	wakeup_flusher_threads(0, WB_REASON_SYNC);
	iterate_supers(sync_inodes_one_sb, NULL);
	iterate_supers(sync_fs_one_sb, &nowait);
	iterate_supers(sync_fs_one_sb, &wait);
	iterate_bdevs(fdatawrite_one_bdev, NULL);
	iterate_bdevs(fdatawait_one_bdev, NULL);
	if (unlikely(laptop_mode))
		laptop_sync_completion();
	return 0;
}
```

Syncfs：根据传入的参数wait/nowait，返回可能失败

```
SYSCALL_DEFINE1(syncfs, int, fd)
{
	struct fd f = fdget(fd);
	struct super_block *sb;
	int ret;

	if (!f.file)
		return -EBADF;
	sb = f.file->f_path.dentry->d_sb;

	down_read(&sb->s_umount);
	ret = sync_filesystem(sb);
	up_read(&sb->s_umount);

	fdput(f);
	return ret;
}
int sync_filesystem(struct super_block *sb)
{
	int ret;

	/*
	 * We need to be protected against the filesystem going from
	 * r/o to r/w or vice versa.
	 */
	WARN_ON(!rwsem_is_locked(&sb->s_umount));

	/*
	 * No point in syncing out anything if the filesystem is read-only.
	 */
	if (sb->s_flags & MS_RDONLY)
		return 0;

	ret = __sync_filesystem(sb, 0);
	if (ret < 0)
		return ret;
	return __sync_filesystem(sb, 1);
}
static int __sync_filesystem(struct super_block *sb, int wait)
{
	if (wait)
		sync_inodes_sb(sb);
	else
		writeback_inodes_sb(sb, WB_REASON_SYNC);

	if (sb->s_op->sync_fs)
		sb->s_op->sync_fs(sb, wait);
	return __sync_blockdev(sb->s_bdev, wait);
}
```

Fsync：等待文件io完成落盘后返回

```
SYSCALL_DEFINE1(fsync, unsigned int, fd)
{
	return do_fsync(fd, 0);
}
```

fdatasync：等待文件的数据io完成落盘后返回

```
SYSCALL_DEFINE1(fdatasync, unsigned int, fd)
{
	return do_fsync(fd, 1);
}

int vfs_fsync(struct file *file, int datasync)
{
	return vfs_fsync_range(file, 0, LLONG_MAX, datasync);
}
```

```
static int do_fsync(unsigned int fd, int datasync)
{
	struct fd f = fdget(fd);
	int ret = -EBADF;

	if (f.file) {
		ret = vfs_fsync(f.file, datasync);
		fdput(f);
	}
	return ret;
}
```

sync_file_range：刷文件的一部分

Since glibc 2.2.2, the Linux prototype for **sync**() is as listed above, following the various standards. In glibc 2.2.1 and earlier, it was "int sync(void)", and **sync**() always returned 0.
According to the standard specification (e.g., POSIX.1-2001), **sync**() schedules the writes, but may return before the actual writing is done. However Linux waits for I/O completions, and thus **sync**() or **syncfs**() provide the same guarantees as fsync called on every file in the system or filesystem respectively.

BUGS

top

Before version 1.3.20 Linux did not wait for I/O to complete before returning.

来自 <http://man7.org/linux/man-pages/man2/sync.2.html>