Whither FileHandle?

This thread has been locked by a moderator.

596

I'm not a big fan of `FileHandle` (`NSFileHandle` in Objective-C) and I wrote this post to explain why.

Note This post focuses on macOS but these problems apply to all Apple platforms.

Performance

Every `FileHandle` read or write call results is a `read` or `write` system call. If you want to read a file one byte at a time, `FileHandle` is not the API for you.

IMPORTANT The async/await read support we added in macOS 12 (currently in beta) fixes this on the read side.

Error Handling

In its original incarnation, `FileHandle` responded to errors by throwing an Objective-C exception. Thus, to use it safely you had to wrap every call with an Objective-C method that catches that exception and returns it as an `NSError` [1].

This was fixed in macOS 10.15, which added new methods that return errors using Cocoa's standard pattern. For example, `readData(ofLength:)` has been deprecated in favour of `read(upToCount:)`, where the latter returns an error rather than throwing an Objective-C exception.

Note Swift code didn't get this fix until 10.15.4.

[1] You may need to write the wrapper in MRR rather than ARC because ARC is not, in general, exception safe. However, I must admit that I've not looked into that aspect of this issue in depth.

Async I/O

`FileHandle` has many different flavours of async I/O support. On the read side this includes APIs based on notifications, a readability handler, and async/await. All of these have their challenges:

- The notification-based APIs require you to have an `NSObject` subclass available.
- The readability handler is called on an unspecified serial queue, which makes it hard to coordinate work with other queues. You can get around this by setting the readability handler to `nil` and then dispatching over to another queue, but that starts getting ugly.
- The async/await API only works on macOS 12.

On the write side things are worse:

- There is no async/await support on the write side (r. 82768702).
- As with the read side, the notification-based API requires you to have an `NSObject` subclass available.
- Both the notification and writeability handler APIs require you to manually enable non-blocking I/O to avoid accidentally blocking. You then have to use the low-level `write` system call to write your data because none of the existing write methods are capable of return info about short writes (r. 82768669).

IMO it's better to avoid the async I/O support in `FileHandle`. My preferred alternative is Dispatch I/O.

The Pipes, The Pipes Are Crashing!

A common use of `FileHandle` is to manage the pipes connected to a child process started using `Process` (`NSTask` in Objective-C). Unfortunately it does not do this job well. Part of this is the poor async I/O support discussed above. A more serious problem relates to `SIGPIPE`.

If you write to a pipe whose remote end has closed, the OS raises a `SIGPIPE` signal whose default disposition is to terminate your process. `FileHandle` does not take any special care to handle this, even if you create your file handles using `Pipe` (r. 82757424). If you use `FileHandle` to manage a pipe manually, set `F_SETNOSIGPIPE` on the file descriptor to avoid this termination. Once you set this, a write will fail with `EPIPE` rather than raising `SIGPIPE`. Yay!

This also applies when you use `FileHandle` to manage a socket, although in that case you use the `SO_NOSIGPIPE` socket option to disable the `SIGPIPE`.

If you have any questions or comments about the above, put them in a new thread. Make sure to tag that thread with *Foundation* and *Files and Storage* so that I see it.

Share and Enjoy
—
Quinn "The Eskimo!" @ Developer Technical Support @ Apple
`let myEmail = "eskimo" + "1" + "@" + "apple.com"`

Foundation Files and Storage

Reply

Posted 1 year ago by eskimo

Add a Comment