NFuse: NFS in User Land

Mohammad Yazdani
University of Waterloo

Abstract

Overall Architecture of the Filesystem

Superblocks and Inodes: NFuse follows the traditional Unix filesystem style of a superblock and inodes. Upon the first boot of the filesystem, the superblock is created and written to disk, and then the filesystem starts.

FObjects: Encompass inodes represent each file/directory in the filesystem. Upon any NFS create request, a new FObject is created, and the inode is written to the filesystem transaction buffer.

Journaling: To handle failures, NFuse uses journaling by synchronously writing every attempt of an operation to the disks. If a journal entry is not written, then the attempt associated with that entry and all attempt with event IDs higher than that attempt is considered invalid. System reports an error in case of journal failure and allows for retry.

Transaction Buffer: Besides the creation of the superblock and the filesystem journal, every I/O operation is queued in the transaction buffer. This buffer is a queue of IOInstr objects which specify an operation, offset, count, and a FEvent object which contains an event ID. Upon a commit/flush, all operations in the transaction buffer will be ordered based on Event Id and will be flushed to disk. If a failure in I/O occurs, the failing item and the rest of the queue will remain intact, and the flush operation concludes an error.

Remote calls and gRPC

Why gRPC? Because of the advantages in latency, bandwidth and ease of integration. In P1 a comparison of different communication packages was made, and for the use of a filesystem, with variable filesizes and need for low latency, gRPC becomes a good choice. Also since the filesystem code is in C/C++ and calls between client and server follow a consistent signature (explained in next section), setting up gRPC would be easier than the development of a TCP stack or a messaging system.

Client-Server Messaging

Consistent Signature: Every call between the client and server has the following properties: - ID: The identifier of the call. - Opcode: Specifies any information such as handler, flag, or status. - Data: The data being sent with this call. - Tags: This field is a string or rather a char buffer of data structured similarly to multiboot tags. The receiving end of the call can parse the tags to get the options specified with this call. For example, if the client sends a write request, the handler on the server-side expects at least two tags: - InitTag struct which specifies the number of tags to be read - IOTag struct which specifies file handler (Inode number), offset and count.

Persistence

Superblock: The server program reads the first N bytes of the mount area dedicated to superblock. After reading the superblock, it can point to the individual Inodes and other filesystem data structures.

Failure

Journaling: Upon any failure or rather crash since this is a non-Byzantine context, the system will rely on the journal as a source of truth. The journal is rolled back until the first committed entry, and then the attempted operations are restarted, before the normal boot of the filesystem.

Code available at: https://github.com/mohammad-yazdani/nfuse