POSIX, and What Comes Next

Andreas Dilger, Principal Lustre Architect
POSIX Is *Really Old*

- Original interfaces developed with Unix in the 1970s
  - Fragmentation as Unix grew and changed in the 1980s
  - Needed a standard for interoperability during "Unix Wars"

- POSIX has been the standard IO interface for decades
  - Hasn't changed significantly in many years, but very widely available
  - Provides the lowest IO common denominator for apps and user tools
  - Consistent behavior means that applications can run everywhere

- Data portability for shared namespace via protocol export (Lustre, NFS, SMB, ...)
  - Avoid data silos by exporting filesystem with (mostly) POSIX semantics to other nodes
  - "Mostly POSIX" can be important, but *different* parts of POSIX needed for different applications

- POSIX consistency can be a bottleneck for some workloads
  - Serialized directory operations, write/read ordering, etc. can slow down performance
Where Are We Now?

► An explosion of new IO interfaces for various special needs
  • New storage systems have their own IO APIs (HDFS, S3, DAOS, ...)
  • Useful for some workloads, but needs significant application investment
  • Specialization ties applications to storage system, loses portability
  • Higher-level libraries abstract new interfaces, but also many libraries

► Leveraging hardware speedups needs optimization
  • Lower storage latency, higher bandwidth
  • Many cores, more and faster network interfaces
  • A rising hardware tide lifts all software, but not equally
    o Leaves "stranded" performance behind
  • Software needs to continually adapt to address bottlenecks
    o Finer-grained threading, locking, concurrency, new interfaces
How To Move Beyond Aging POSIX Standard?

► Embrace and Extend
  • Some tries at HPC extensions (stat_lite(), open_by_handle(), ...)
  • Didn’t make it into official POSIX standard, but were added to Linux

► Linux provides de-facto standard for new interfaces
  • New IO interfaces are being added incrementally
  • Sometimes adopted from other OSes (BSD, Solaris, ...)
  • open_by_handle(), name_to_handle_at(), fallocate(), copy_file_range(), pwritev2()
  • *_at(), statx(), O_TMPFILE, FIEMAP, SEEK_HOLE/DATA, ...

► DAX for memory load-store access to persistent memory
  • Used by SPDK to provide access to NVRAM managed by ext4/XFS

► Asynchronous data AIO/DIO via libaio
  • Originally used by databases, but could be leveraged by any tools with a lot of concurrent IO

► Asynchronous data and metadata operations via io_uring with growing capabilities
  • Provides may POSIX syscall equivalents with completion callbacks, including some metadata syscalls
What Happens in the Future?

► POSIX continues to be the common interface going forward
  • Important for interoperability during "IO Interface Wars"
  • Protects significant investment in developed applications and tools
  • By necessity, most storage systems must also provide a POSIX interface

► Sometimes bottleneck is in implementation, not POSIX
  • Serialized single directory operations is Linux VFS implementation limit

► **API extensions** for apps with special performance needs
  • Specialized interfaces **opt-in** when/where applications need it
  • Easier to relax strong POSIX semantics by request than miss them and cause corruption/bugs
  • Applications can leverage new APIs via common libraries or directly for performance reasons
  • Data continues to be accessible via standard POSIX APIs/tools after creation/processing