Storage Systems

NPTEL Course

Jan 2012

(Lecture 24)

K. Gopinath

Indian Institute of Science

FS semantics

- Mostly POSIX notions
- But not really fixed
  - Many impl flexibilities/dependencies allowed
  - Atomicity of a copy of a 1TB file difficult!
  - try “cp a, b&; cp b, a”
- Often need to support common idioms, even if not in POSIX
  - Shell scripts' expectations

Right specification difficult to define: mmap

- mmap depends on VM: granularity at page level only
- what is the semantics of read/write past eof in the middle of a page?
- what happens with extending writes in presence of concurrent r/w?
- when can an user see the writes (during or after write?)

File Corruption Prob

- Files rapidly accessed on a WinNT file server, intermittent data corruption! Sep96

  esp news server data files & on MP systems
- appln performing a write-extend of a file
- cache manager read ahead thread on current last page (part of larger read); write blocked
- mem manager wakes up write & zeroes last page beyond curr file size & writes new data into page; read also zeroes later!

Record and file locking: cp a, b&; cp b, a

- one rwlock: deadlock! but with rwlock + glock: OK
  - cp a, b: maps a & then write into b from mapping of a
    - mmap not atomic but r/w “atomic”: need rwlock (shared/excl)
    - mmap'ed pages may fault: need to call VOP_GETPAGE
      - Holes in file: allocates atomic; need a lock (glock)
      - Interlocking betw truncate ops and getpage ops: truncate has to prevent getpage from bringing in pages; use glock
    - mmaping: rwlock a; map; rwunlock a;
    - reading: rwlock b; uiomove (getpage a); ...
  - cp b, a:
    - rwlock b; map; rwunlock b;
    - rwlock a; uiomove (getpage b); ...
- If both glock and rwlock the same lock? deadlock!

Locking issues

- a thread locks a resource and calls a lower-level routine on that resource; lower-level routine may also be called by others without locking resource
- deadlock possible if lower-level routine does not know if resource locked
  - unless params passed informing low-level routine about locked resources *but* this is non-modular!
- recursive locks can avoid deadlocks: need owner ID
  - functions deal with their own locking reqs: clean, modular i/f; do not need to worry about what locks callers own
  - Ex: ufs_write handles both writes to files/dirs
    - files: unlocked vnode passed from file tbl entry to ufs_write
    - dirs: vnode passed locked f DNLC/pathname traversal func

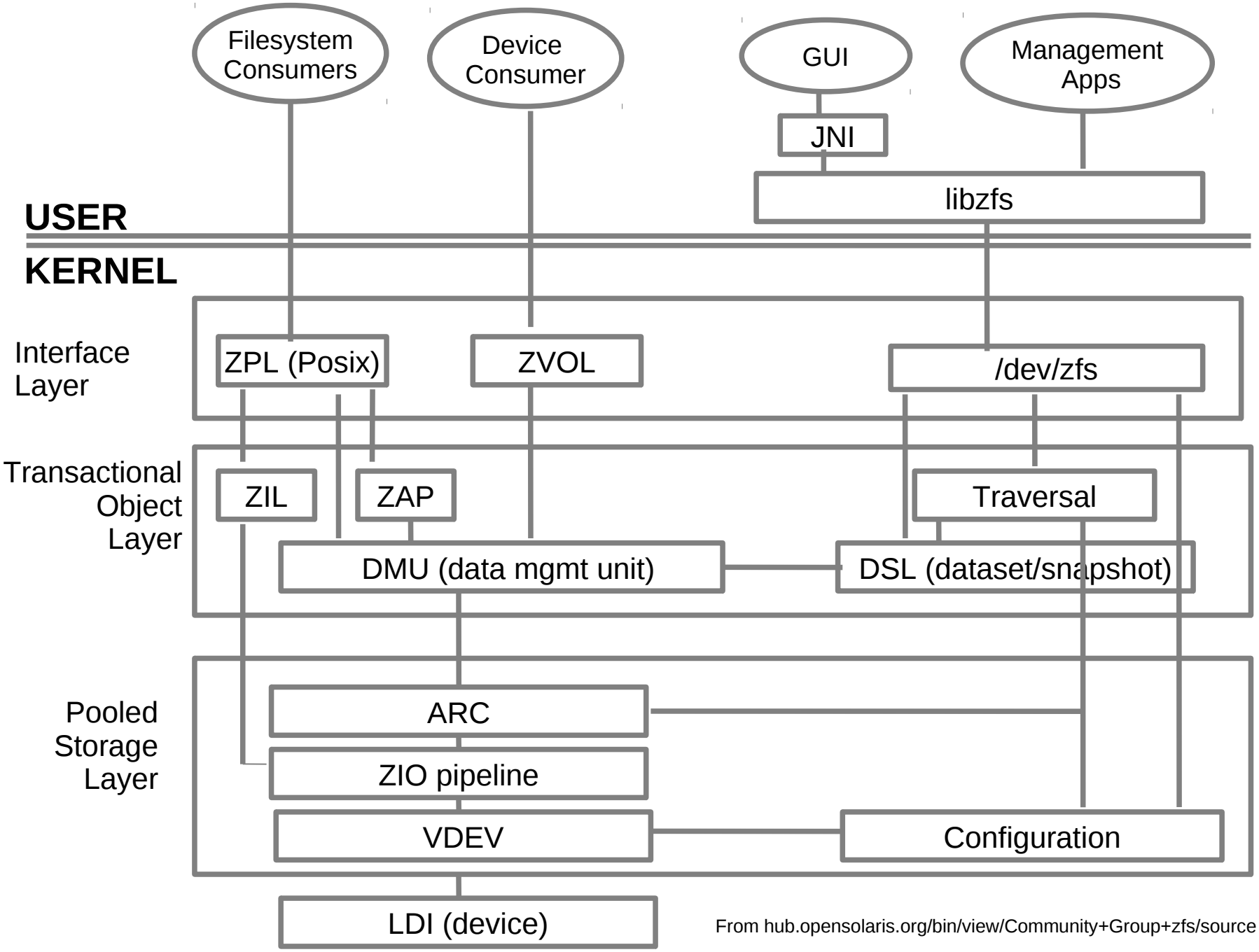
ZFS

- Integrated file system + logical volume manager
  - zfs and zpool: ZPL (ZFS POSIX Layer), DMU (Data Management Unit), and SPA (Storage Pool Allocator)
  - design for fast, reliable storage using cheap, commodity disks
  - copy-on-write transactional object model
    - blocks containing active data never overwritten in place
    - instead, a new block allocated + written, then any metadata blocks referencing it are similarly read, reallocated, and written.
    - To reduce overhead, multiple updates grouped into transaction groups, and an intent log used when synch write semantics required

- Key features
  - data integrity verification against data corruption
    - Each block of data/metadata checksummed and checksum value saved in the pointer to that block (not in the actual block itself)
  - Merkle tree maintained (all thru file system's data hierarchy to root node)
- support for high storage capacities (128-bit FS)
    - zpool made of virtual devices (vdevs) that are constructed from block devices
  - RAID-Z to avoid “write-hole”. Also RAID-Z2/3
  - ARC2 (read caching) and ZIL (ZFS intent log for write caching)
  - snapshots and copy-on-write clones
  - native NFSv4 ACLs, deduplication, encryption, compression

Design

- User level:
  - FS consumer: uses Posix ZFS fs
  - device consumer: uses devices avlbl thru /dev
  - GUI (JNI), Mgmt Apps (both access ker thru libzfs)
    - eg. zpool(1M), zfs(1M)
    - libzfs: unified, object-based mechanism for accessing and manipulating storage pools and filesystems
- Kernel Level:
  - Interface Layer
  - Transactional Object layer
  - Storage Pool Layer



From hub.opensolaris.org/bin/view/Community+Group+zfs/source