Solaris Volume Manager : Kernel Model

Drivers

- md driver is the only “real driver in the system”
- md driver is loaded by the system during svm startup
- Other drivers are subordinates (subdrivers to md)
- Other drivers are loaded and accessed via the md driver
- All access to subdrivers is through the md driver

md devops

- When the md driver is loaded, the _init routine links the md _devops structure into the I/O subsystem exporting these entry points for md
  - mdopen
  - mdclose
  - mdstrategy
  - mdread
  - mdwrite
  - mdioctl
- The devops structure is defined by the DDI/DKI interface

Subdrivers

- md_stripe MD_STRIPE
- md_mirror MD_MIRROR
- md_hotspares MD_HOTSPARES
- md_raid MD_RAID
- md_sp MD_SP
- md_notify MD_NOTIFY
- md_verify MD_VERIFY (in test suites)
- md_trans MD_TRANS (legacy, pass through)

Subdrivers (cont)

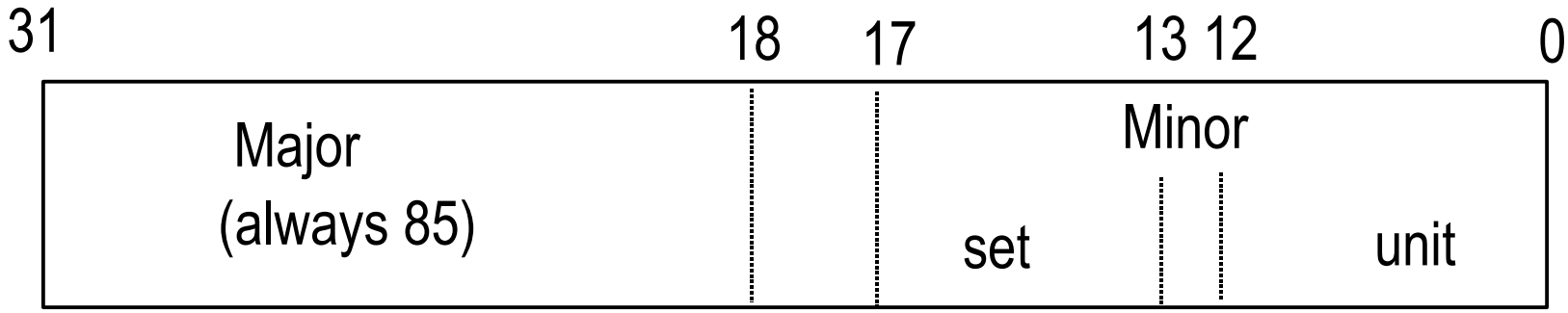
- When a subdriver is needed the md driver loads the corresponding md_ops structure in to the global md_ops array.
- md_ops structures is only used by the md_driver to communicate with the subdrivers
- md_ops structure can be changed

Subdrivers (cont)

- md_ops structures provides entry points into the subdriver. Here are the entry points for stripe_md_ops
  - Stripeopen
  - Stripeclose
  - md_stripe_strategy
  - stripe_ioctl
  - stripe_snarf
  - stripe_halt
  - stripe_imp_set
  - stripe_named_services

dev_t model from Userland

- Upper 14 bits are major number
- Lower 18 bits divided into
  - 5 bits for set number
  - 13 bits for unit number



minor 0x3ffff (262143) is the admin device

dev_t Model from Kernel

- dev_t in kernel is always in native form
- Legacy structures on drive are in 32 bit form
- dev_t is converted from on disk form to native form via md_cmpldev and md_expldev

dev_t examples

85, 262143	/dev/md/admin admin special minor of 0x3ffff
85, 15	/dev/md/dsk/d15 unit 15 in local set
85, 8207	/dev/md/foo/dsk/d15 /dev/md/shared/1/dsk/d15 unit 15 in diskset 1

Use of Admin Minor

- Only accessed via ioctl interface
- Handles administrative commands not directed to a specific subdriver
- Command may be passed to subdriver if md cannot handle the request

Use of Non-Admin (unit number) Minor

- Supports read/write/ioctl interfaces
- Supports buf interface (via strategy)
- Unit number used to determine subdriver
- Request is received by md driver and directed to appropriate subdriver

Ioctl Interface

- Single Threaded Ioctls
  - Uses global ioctl lock bit MD_GBL_IOCTL_LOCK
  - Locks out all ioctls from ALL disksets
  - If lots of threads in md_ioctl then the thread holding the global lock is stuck

Ioctl Interface (cont)

- Multi Threaded Ioctls
  - NOT locked out by global ioctl lock
  - Count kept in global md_mtioctl_cnt
  - Introduced for multi-owner disksets (oban)
  - List of mt ioctls found in routine is_mt_ioctl()

Ioctl Interface (cont)

- Ioctl Locks
  - mdioctl inits an IOLOCK (lockp) for each ioctl call
  - Some routines are entered via ioctl and non-ioctl (read/write/strategy) interfaces
  - Will only have non-null lockp if called via ioctl
  - Important for multi-owner disksets when all locks must be dropped or a deadlock will occur.

Ioctl Interface (cont)

- IOLOCK (lockp) filled in with unit and lock type during:
  - md_ioctl_readerlock/writerlock
  - md_ioctl_io_lock
  - md_array_reader/writer
  - md_ioctl_openclose
- Can be used when debugging to locate unit/lock type

Read/Write Interface

- Reads and Writes are converted to buf interface and sent to mdstrategy via physio (sync) or aphysio (async).
- Subdrivers do not support read/write directly

Strategy Interface

- Calls to mdstrategy are coming from within the kernel. Either from md (md*read/md*write) or from another module such as a file system.
- mdstrategy calls the subdriver strategy routine
- mdstrategy and subdriver strategy routines must call either biodone or md_biodone when I/O has finished

md_biodone

- For disksets the subdriver strategy routines increment a set-wide outstanding I/O count: md_set_io[setno].io_cnt
- After the count has been incremented, the subdriver must call md_biodone when the I/O finished.
- md_biodone will:
  - Always call biodone
  - Decrement the io_cnt for a diskset

Basic Subdriver Strategy

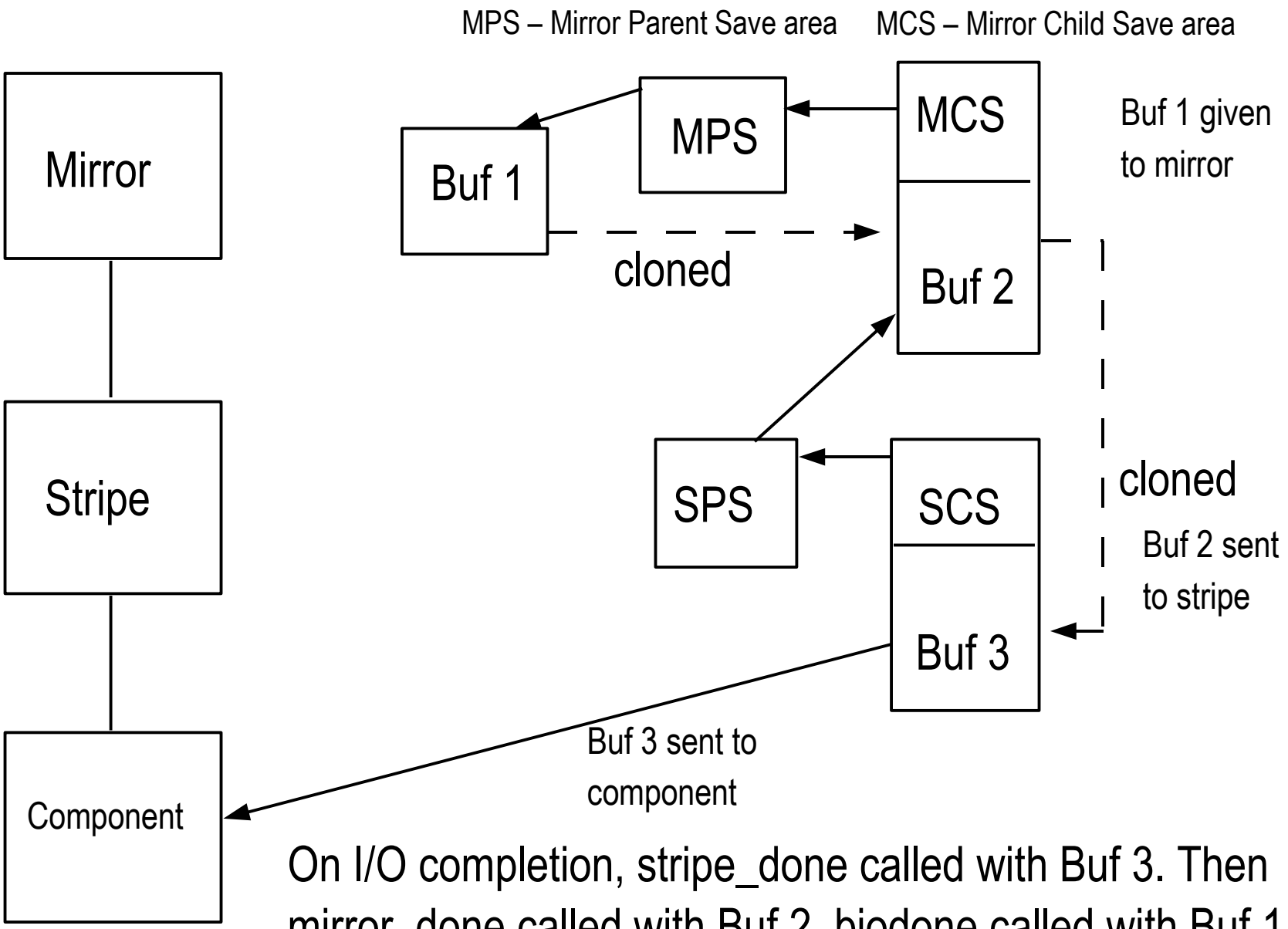
- Subdriver allocates parent and child save structures
- Subdriver calls md_bioclone to clone buf in child save area
- Cloned buf has devt, offset, blkno, blkcnt set for component
- Cloned buf has iodone routine set to subdriver's done routine

Basic Subdriver Strategy (cont)

- Parent and child save area loaded with info
- Cloned buf sent to component driver's strategy routine
- Original buf not tracked once issued to component driver
  - Good for performance, bad for debugging

Basic Subdriver Strategy (cont)

- Subdriver iodone routine called when I/O has finished.
- Use “child save” space in front of cloned buffer to recover cloned buf and parent save area
- Set appropriate return values in original buf
- Call md_biodone with original buf



On I/O completion, stripe_done called with Buf 3. Then mirror_done called with Buf 2, biodone called with Buf 1.

Subdriver Locks

- md_unit_readerlock – allows multiple readers
- md_unit_writerlock – only allows 1 writer and no readers
- md_unit_openclose – held during open and close of unit
- Only used by RAID
  - md_io_readerlock – grab io lock and reader lock
  - md_io_writerlock – grab io lock and writer lock

Stripe Subdriver

- I/O to stripe may cause multiple requests to component driver if original request spans more than 1 component or partition
- Error from component passed back to caller
- No hotsparing occurs in the stripe subdriver

Softpart Subdriver

- I/O to softpart may cause multiple requests to component driver if original request crosses an extent boundary
- Error from component passed back to caller
- No hotsparing occurs in the softpart subdriver
- Softpart information stored on-disk with the data
  - Name of sp (setname and metadevice name)
  - Number of extents and offsets for sp
  - Can be recovered from on-disk information

Raid Subdriver

- Each write causes an additional write to the parity device and an additional write to the prewrite log
- A read may cause additional reads if the component has failed and the data needs to be regenerated from the parity device
- Component failure can cause hotspare operation
- Data sent to hotspare is regenerated from remaining components
- Cannot withstand failure to 2 components

Raid Subdriver (cont)

- A component in a RAID metadevice can be in these states:
  - CS_OKAY
  - CS_ERRED
  - CS_RESYNC
  - CS_LAST_ERRED

Mirror Subdriver

- Each write may cause a write to the optimized resync records in 2 mddbs that contain a dirty region list.
- Each write causes a write to all submirrors.
- A read may cause additional reads if a component has failed and the data needs to be read from another submirror.

Mirror Subdriver (cont)

- Component failure can cause hotspare operation
- Data sent to hotspare is copied from a remaining submirror
- Cannot withstand failure of all components

Mirror Subdriver (cont)

- A component in a MIRROR metadevice can be in these states:
  - CS_OKAY
  - CS_ERRED
  - CS_RESYNC
  - CS_LAST_ERRED

Mirror Resync

- Synchronizes data between all submirrors and components
- Data on all submirrors may not be the latest data (if a node panicked during a write), but must be the same after the resyncs

Mirror Resync (cont)

- Reads and writes occur during the mirror resyncs
- Resync code locks down a resync region and writes will be delayed until the resync moves to the next window

Mirror Resync (cont)

- Component
  - Used to resync new component of submirror
  - Resync to hotspare
  - Resync to replaced component
- Submirror
  - Used to resync a new submirror
  - Attach of new submirror

Mirror Resync (cont)

- Optimized
  - Used to quickly sync mirror using dirty regions
  - Used if node panicked during write to submirrors
  - Used if offlined submirror is onlined
  - Uses optimized resync records in 2 mddbs
  - If both resync records are unreadable, then do full submirror resync
- ABR
  - Don't update dirty regions since application will handle the optimized resync situation

Failfast

- Disk driver can take a LONG time to fail each queued I/O
- I/Os with the FAILFAST flag are retried for only 2 minutes and then all queued failfast requests are marked as failed after the first failure is seen to that drive
- Mirror subdriver uses failfast capability if more than 1 copy of the data is available

Daemon Queues

- Used to queue up requests to be handled at a later time or by a different thread
- Queue contains linked list of requests
- Be sure to check queues when looking for stuck I/O

Daemon Queues (cont)

- Many queues
  - md_mstr_daemon – mirror/raid timeout requests
  - md_mhs_daemon – mirror hotspare requests
  - md_done_daemon – done requests
  - md_mirror_daemon – mirror owner requests
  - md_mirror_rs_daemon – mirror resync done request
  - md_mirror_io_daemon – mirror owner i/o request
  - md_ff_daemonq – failfast request
  - md_hs_daemon – raid hotspare request
  - md_sp_daemon – softpart error request

Solaris Volume Manager : Kernel Model