GEOM Tutorial

Poul-Henning Kamp

phk@FreeBSD.org

(Recovering LEGO addict)

Outline

- Background and analysis.
- The local architectural scenery
- GEOM fundamentals.
- (tea break)
- Slicers (not a word about libdisk!)
- Tales of the unexpected.
- Q/A etc.

Shakespeare (ver. 1.08)

To architect or not to architect ?
That is the question: 'tis nobler in the kernel to
have limitations of hard coded policies,
Or to design against a mess of hacks,
and by fixing them ? add new mistakes.

a bit of UNIX history

- Disk partitioning came to UNIX very early.
- Hard coded in the disk device drivers.
- An architecturally clean solution:
  - Drivers already have abstractions for multiple devices.
  - Hard coded means no admin tools needed.
  - No meta-data modification problem.

Struct buf and all that.

- To do disk-I/O:
  - Get a buffer
  - Set the fields right.
  - Call strategy().
- All disk-I/O happens in/on struct buf.
- Struct buf is the protocol.
- Struct buf is also a strictly defined cache.

Progress...

- The hard coded table became a bother.
  - Put partition table in magic sector.
  - Read it once, at boot.
- Still architecturally clean solution.

... is overrated ...

- On the fly modification.
  - Add ioctls() to modify label on the fly.
  - Add admin tools to do so.
  - Some details in the corners hacked around.
- Crumbling of architecture.
  - magic 'c' partition.
  - special “write-protect label” ioctls.
  - boot code stored inside file system partitions.

...but seldom...

- Arrival of PC architecture adds more hacks.
  - label inside partially trust-worthy MBR slice.
  - magic 'd' partition as “really entire disk”
  - tools to fiddle MBRs.
  - hacks to supply MBR distrust workaround.
  - Dangerously Dedicated and all that...
- Architecture goes underground.

... goes too far.

- Code cleanup adds pseudo-quasi-crypto-generic-oid two-level slice/partitioning code.
  - Codifies the two-level structure of IBM/pc in MI code.
  - Implements "compat slice" to allow purists to ignore MBR (at their peril)
  - Uses absolute offsets in second level label data.
- Architecture for the past instead of the future.

Pressure from the side.

- CCD stripe/mirror "pseudo" device driver.
  - Not "pseudo" at all.
  - Stealth use of buffer cache API.
  - Fortunately no meta-data.
- Vinum
  - CCD on steroids. Veritas aspirations.
- RaidFrame
  - Research RAID engine.

Features...

- All US bank-notes are same size and green.
- Originally this was a hack: Cheap, efficient for production.
- Turned into a feature when people started to depend on it: wallets, counting machines vending machines.
- Feature has a large and addicted user base.

... and misfeatures.

- Feature becomes misfeature:
  - trivially simple to counterfeit greenbacks.
- Drastic alterations impossible, the addicted user-base would scream and yell.
  - and they can, unfortunately, afford politics.
- Countermeasures must "fit in format"
  - not efficient --you need a microscope.

Our features...

- CCD was a hack.
- For the lack of something better, people started to depend on it.
- s/hack/feature/
- People wanted more.
- ~~"Hang on while we fix our architecture."~~
- "Sure, here's Vinum and RaidFrame!"

Architecture is hard...

- Lets go hacking!
- We stand on the shoulders of giants.
- We tend to forget that too often.
- "Infrastructure" is the key to high quality in any program.
- It's often more important to remove than to add to a program.

Sheep vs. Wolves

- The ability for Open Source to compete depends critically on our ability to sustain a viable infrastructure.
- The much famed BSD network stack has already been eclipsed by Linux: (Sel-Ack, NewReno, etc etc)
- Forwards compatibility must never give way to backwards compatibility.

Sheep vs. Wolves

- Some face even bigger problems than us:
  - Solaris still reserves "alternate cylinders"
    - Not sure what would break, dare not remove.
  - Some have heavy legacy code tied in:
    - Veritas Volume Manager for instance.
- Some have far less:
  - We're Microsoft, we decide the "standards".

GEOM does what ?

- Sits between SPECFS and device-drivers
- Provides framework for:
  - Arbitrary transformations of I/O requests.
  - Collection of statistics.
  - Disk-sort like optimizations.

The GEOM design envelope.

- Modular.
- Freely stackable.
- Auto discovery.
- Directed Configuration.
- POLA
- DWIM
- No unwarranted politics.

“Modular”

- You cannot define a new transformation and insert it into Veritas volume manager, AIX LVM, Vinum or RaidFrame.
- They are all monolithic and closed.
  - “A quaint feature from the seventies”.
- We want to do better than that...

Auto discovery.

- Classes allowed to “automagically” respond to detectable clues.
  - Typically reacts to on-disk meta-data.
    - MBR, disklabel etc
  - Could also be other types of stimuli.

Directed configuration

- "root is always right"
  -- the kernel.
- Root should always be able to say "You may think it sounds stupid, but I want it!"
- ...as long as it does not compromise kernel integrity.

POLA

- Principle of Least Astonishment.
- Pola is not the same as
  "retain 1.0 compatibility at any cost!"
- Very hard to describe or codify, but intuitively obvious when violated.

DWIM

- Do What I Mean.
- Have sensible defaults.
- Make interfaces versatile but precise.
- Make sure interfaces have the right granularity.
- Be liberal to input, conservative in output.
- And a total bastard to the programmers.

Say again ?

- I detest people who take short-cuts rather than do things right, because they leave shit for the rest of us to clean up.
- GEOM is fascist in certain ways, to prevent certain "obvious" hacks.
  - Try to sleep in the I/O path -> panic.
  - Lots of KASSERTS.
  - Etc.

No unwarranted Policies.

- “FreeBSD, tools, not policies”.
- We are not in the business of telling people how they should do their work.
  - Unlike some operating systems I can mention.
- UNIX is a success because it's a tool chest.
  - Lets keep it a success.

No unwarranted Policies.

- Leave maximal flexibility to the admin.
- Don't restrict use based on your:
  - High moral ground posturing
    - “Telnet is insecure, REMOVE IT!”
  - Unfounded theories
    - More or less anything Terry ever said.
  - Weak assumptions
    - “Heck nobody would ever do that!”

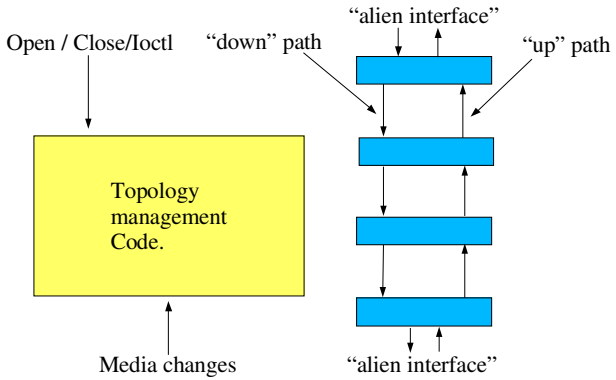
Freely stackable.

- Put your transformations in the order you like.
  - Mirror ad0 + ad1, partition the result.
  - vs
  - Partition ad0 and ad1, mirror ad0a+ad1a, ad0b+ad1b, ad0c+ad1c, ad0d+ad1d ...
- Strictly defined interfaces between classes.

Technical requirements.

- SMPng style.
  - Giant-less.
  - Good granularity.
  - Strict but sensible locking.
- Break the kernel stack depth.
  - a class can be complex, a stack of classes can be very complex.
- Efficient.

GEOM, the big overview.



GEOM terminology.

- "A transformation"
  - The concept of a particular way to modify I/O requests.
    - Partitioning (BSD, MBR, GPT, PC98...).
    - Mirroring
    - Striping
    - RAID-5
    - Integrity checking
    - Redundant path management.

GEOM terminology.

- "A class"
  - An implementation of a particular transformation.
    - MBR (partitioning)
    - BSD (ditto)
    - Mirroring
    - RAID-5
    - ...

GEOM terminology.

- "A geom" (NB: lower case)
  - An instance of a class.
    - "the MBR which partitions the ad0 device"
    - "the BSD which partitions the ad0s1 device"
    - "the MIRROR which mirrors the ad2 and ad3 devices"
    - ...

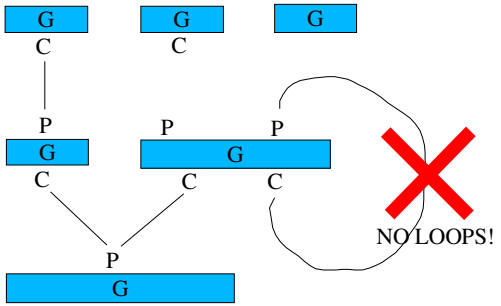
GEOM terminology.

- "A Provider"
  - A service point offered by a geom.
  - Corresponds loosely to "/dev entry"
    - ad0
    - ad0s1
    - ad0s1a
    - ad0.ad1.mirror

GEOM terminology.

- "A consumer"
  - The hook which a geom attach to a provider.
  - name-less, but not anonymous.

GEOM topology.



Topology limits:

- A geom can have 0..N consumers
- A geom can have 0..N providers.
- A consumer can be attached to a single provider.
- A provider can have many consumers attached.
- Topology must be a strictly directed graph.
  - No loops allowed.

I/O path.

- Requests are contained in “struct bio”.
- A request is not transitive.
  - Clone it and pass the clone down.
- “start” entry point in geom used to schedule.
- bio->bio_done used to signal completion.
- dedicated non-sleepable threads for pushing bios around.

I/O efficiency.

- Cannot sleep in up/down path
  - Enforced with hidden mutex.
- Don't do CPU heavy tasks in the up/down paths, use separate kthreads.
- More parallel threads possible.
  - Makes no sense for NCPU < 4

I/O locking.

- Mutex on individual bio queues.
- Bio request scheduled on consumer.
  - Fails if not attached and open(ed enough).
- Bio records “from + to”.
- Bio reply follows recorded “to->from” path
  - Technically it will be possible to answer after path has been removed. (do not try this at home!)

Topology locking.

- The “topology lock”
  - Must be held to change the topology.
  - Must be held during open/close processing.
  - Not needed for I/O processing.
  - Doesn't stop I/O processing.
- Single “giantissimo” lock warranted by low frequency of use.

Class primitives.

- Create Class
  - Adds class to list of classes.
- Destroy Class
  - Fails if class in use.

Geom primitives

- Create geom.
  - Geom belongs to specified class.
- Destroy geom
  - Fails if geom has consumers
  - Fails if geom has providers.

Provider primitives.

- Create provider.
  - Provider belongs to specified geom.
- Set provider error code.
  - Specify error code to stop all I/O.
- Orphan provider.
  - Tell consumers to bugger off. (more later).
- Destroy provider
  - Fails if in use.

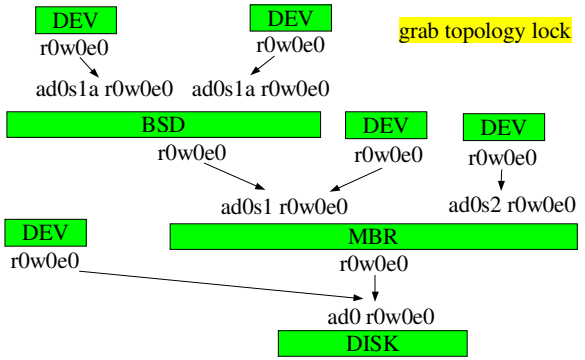
Consumer primitives.

- Create consumer
  - Consumer belongs to specified geom.
- Attach consumer to specified provider
- Change access counts of consumer.
  - Fails if not permitted or not attached.
- Detach
  - Fails if non-zero access counts.
- Destroy (Fails if attached)

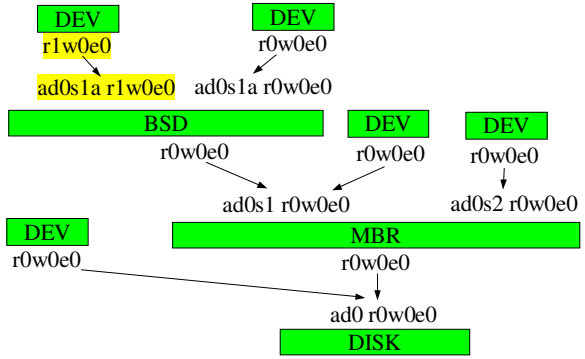
Access counts.

- Access is tracked as three reference counts:
  - Read gives read access. Practically always granted.
  - Write gives write access.
  - Exclusive prevents others write access.
- Consumer and providers have associated counts.
- Providers count is the sum of all attached consumers counts.

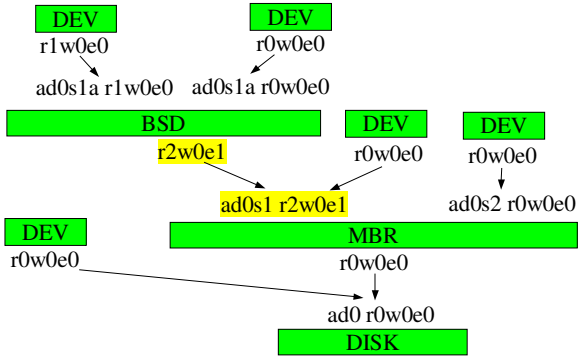
How access counts work (1)



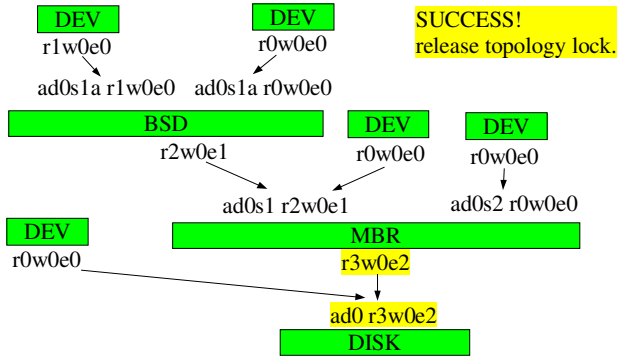
How access counts work (2)



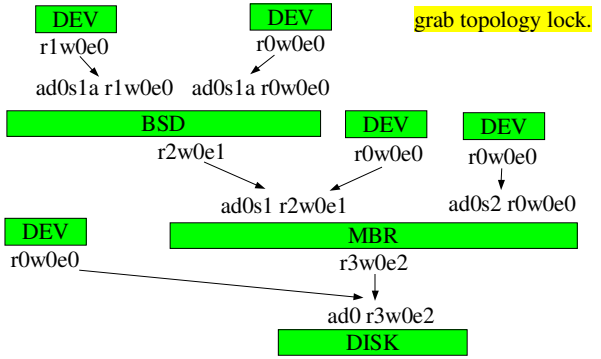
How access counts work (3)



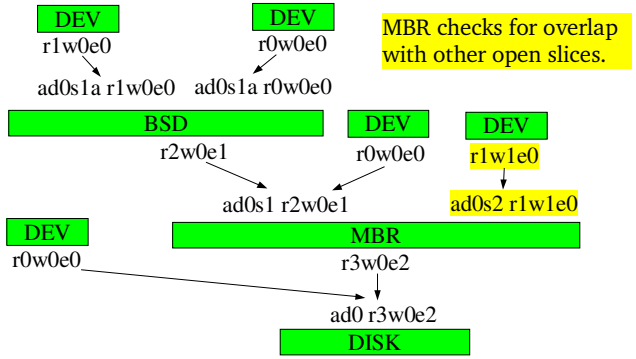
How access counts work (4)



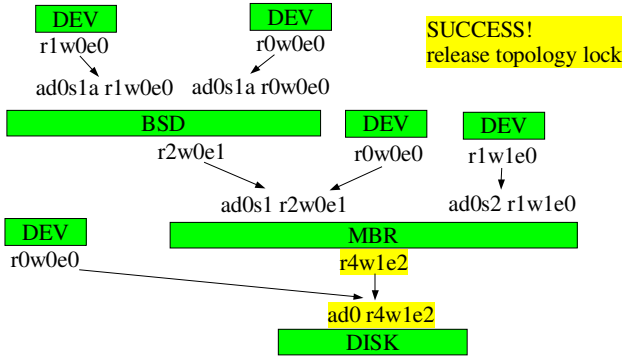
How access counts work (5)



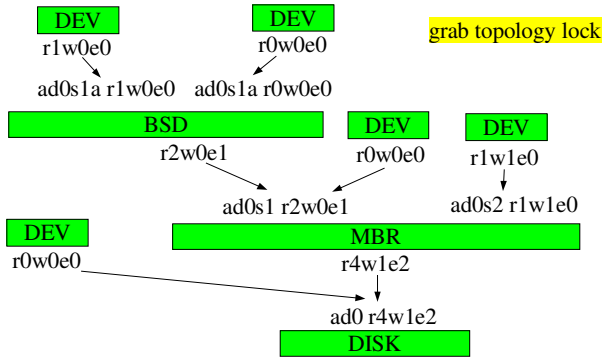
How access counts work (6)



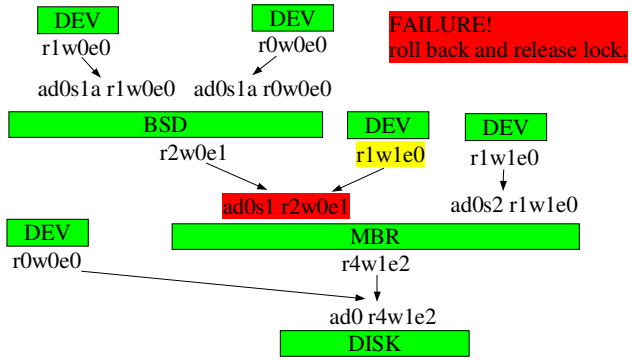
How access counts work (7)



How access counts work (8)



How access counts work (9)



GEOM ahead of the kernel.

- Kernel didn't used to provide strong access checks at the disk-IO level.
- Primitives insufficient to express R/W/E policy fully.
- File systems sloppy with handling even what is supported.
  - mount r/o: => open r/o
  - remount r/w => no reopen to r/w mode.

Events and all that.

- GEOM has an internal job-queue for executing auto discovery and other housekeeping.
- Events posted on a queue.
- Dedicated event thread grabs topology lock, executes event releases lock.
- Event queue protected by event lock.

user land and events.

- All user land operations which need topology lock must wait for empty event queue.
  - open/close/ioctl
- Explicit “process all events” calls may be needed in class code.
- (This can maybe be improved in a generic fashion)

“New Class” event.

- Posted when a class is added.
- Results in the class being offered a chance to “taste” all current providers in the system.

“New Provider” event.

- Posted when provider is created.
- Posted when a provider write access count goes to zero.
- All classes offered a chance to taste the provider.

“Spoiled” event.

- Posted when a provider gets a non-zero write access count.
- All attached consumers, except the guilty party, notified.
- (More on this later).

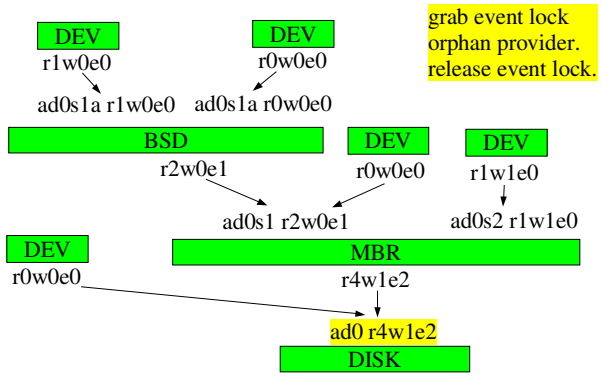
“Orphan” event..

- Devices disappear without notice.
- That's hardware for you...
- Not nice from a UNIX philosophy.
- But we have to cope...

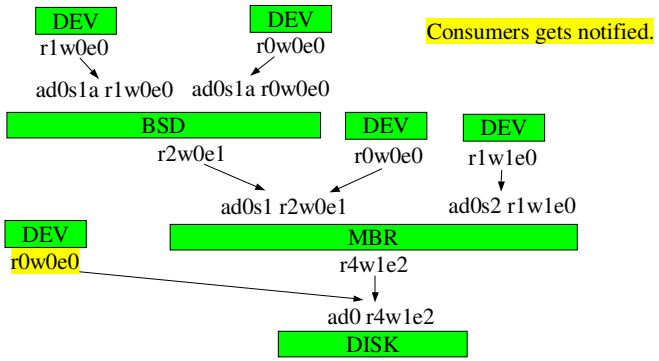
“Orphan” event..

- A provider can be “orphaned” by its geom.
  - All future I/O requests fail.
  - In-transit I/O requests can complete
  - consumers get notified.
  - consumers expected to zero access counts and detach.
  - only then can the provider be destroyed.

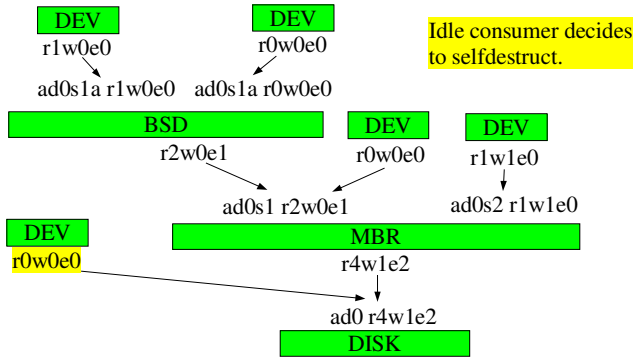
How orphaning work (1)



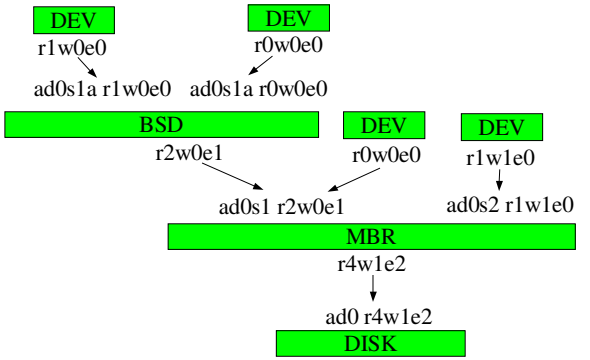
How orphaning work (2)



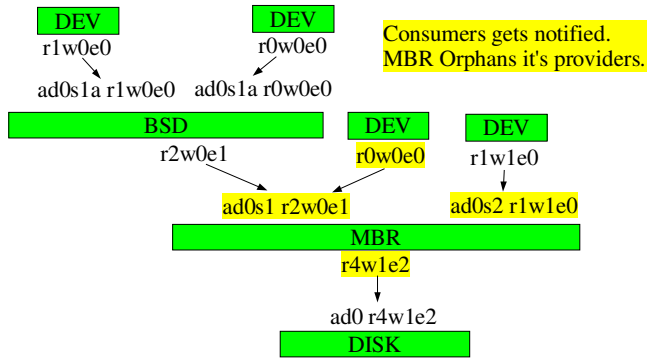
How orphaning work (3)



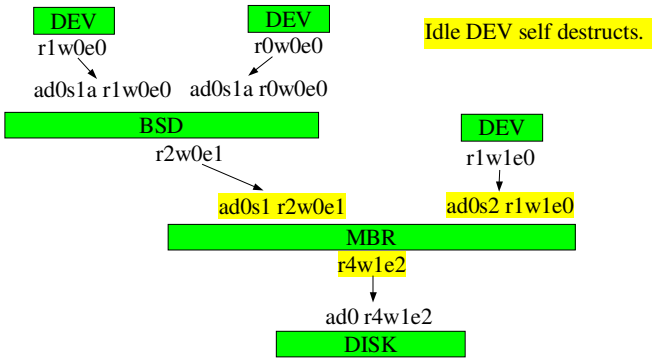
How orphaning work (4)



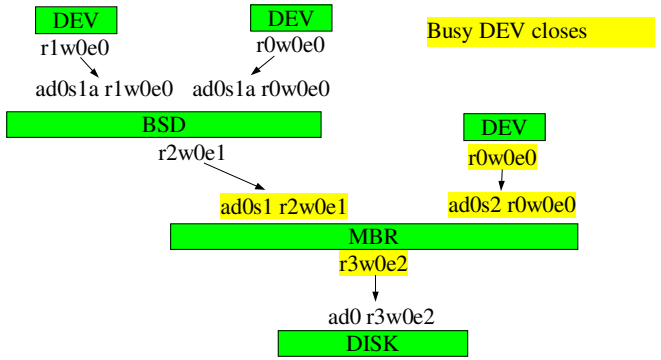
How orphaning work (5)



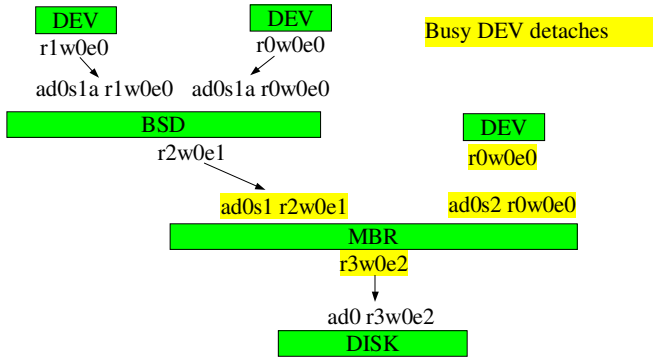
How orphaning work (6)



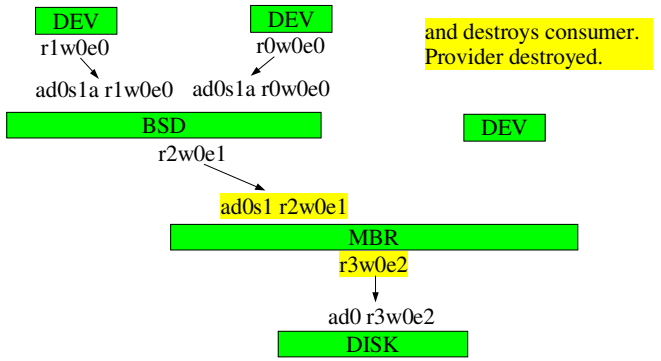
How orphaning work (7)



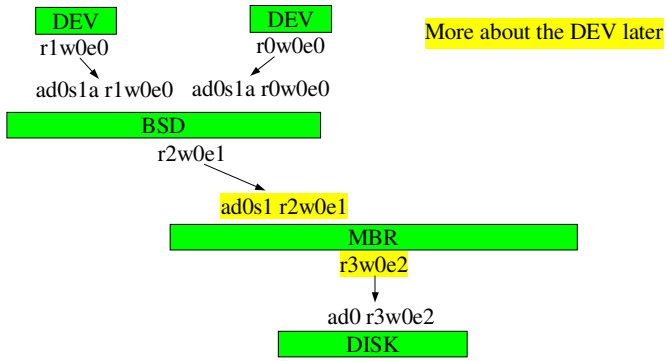
How orphaning work (8)



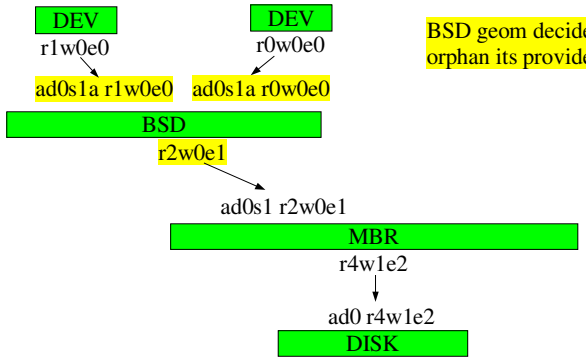
How orphaning work (9)



How orphaning work (10)

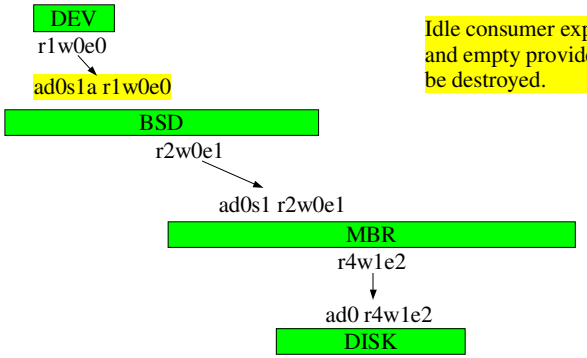


How orphaning work (11)



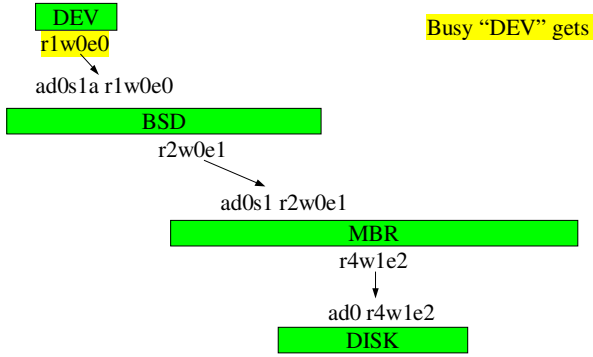
BSD geom decides to orphan its providers.

How orphaning work (12)



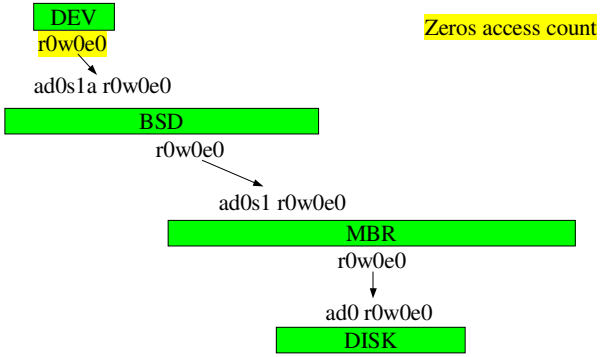
Idle consumer explodes and empty provider can be destroyed.

How orphaning work (13)



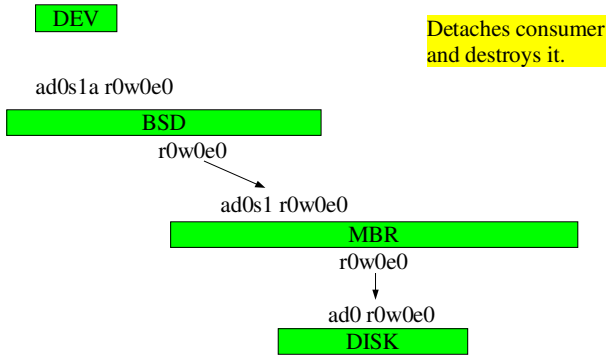
Busy “DEV” gets notified

How orphaning work (14)

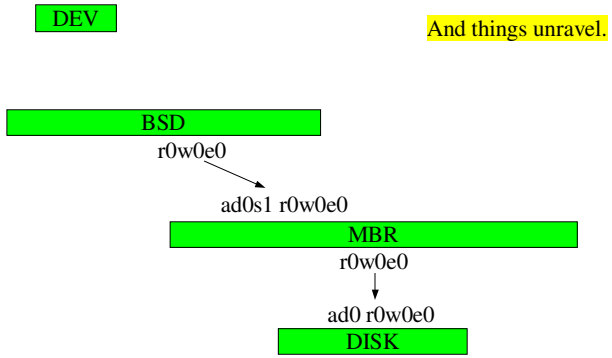


Zeros access count

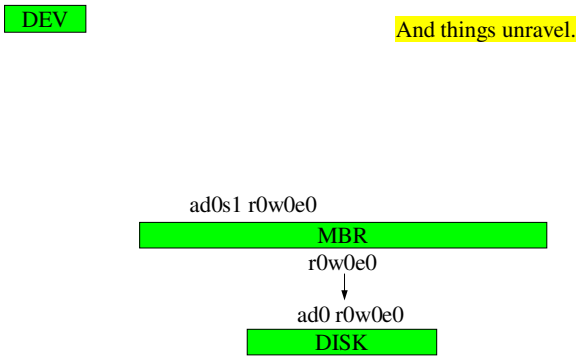
How orphaning work (15)



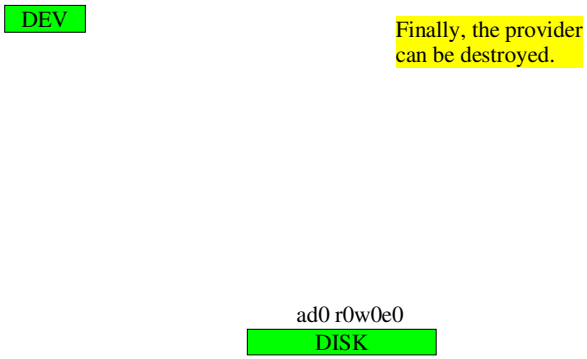
How orphaning work (16)



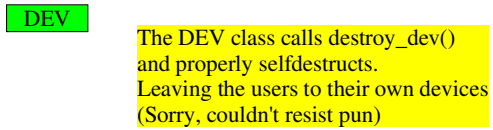
How orphaning work (17)



How orphaning work (18)



How orphaning work (19)



Special GEOM classes.

- There are no special GEOM classes.
- All GEOM classes are treated the same.
- ... But not all GEOM classes have the same kind of job.
  - "DISK" class talks to disk device drivers.
  - "DEV" class talks to dev_t/SPECFS/DEVFS.
  - "GEOMCTL" offers management interface.

the DISK geom class.

- Upper side interface: GEOM
- Lower side interface: "disk minilayer"
  - disk_create().
    - do magic necessary for disk device-driver.
    - Create a provider.
  - disk_destroy().
    - Orphan provider.
    - do various magic for the disk device-driver.
    - self-destruct when possible.

The DEV geom class.

- Lower side interface: geom consumer.
  - Attaches to anything taste presents to it.
- Upper side: disk device-driver.
  - Calls make_dev() with suitable args.
- When Orphaned:
  - Calls destroy_dev()
  - Selfdestructs.

Would it be possible...

- To write a GEOM class to sit on top of the network ?
- To give disk device drivers a native GEOM interface instead of using the DISK class ?
- To ... ?
- YES, Geom classes are very very general.

Spoiling

- A new disk arrives: /dev/da0
- A NEW_PROVIDER event gets posted.
- All classes gets to taste the disk.
- BSD finds a disklabel and attaches.
- User does: dd if=/dev/zero of=/dev/da0
- The disklabel which configured the BSD is gone, and the BSD geom needs to know.

Spoiling (1)

- A class which relies on on-disk meta data will set exclusive bit if it is open in any way.
- This prevents opens which could overwrite the meta-data without us noticing.

Spoiling (2)

- When a provider is opened for writing first time (write access count goes non-zero):
  - Post spoil event on all attached consumers except they guilty party.
  - Consumers which rely on meta data, are obviously closed (otherwise you couldn't open for writing) and they self destruct.

Spoiling (3)

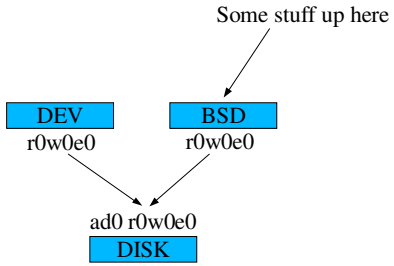
- When the provider is closed (ie: write access count goes to zero)
  - NEW_PROVIDER event posted on provider.
  - All classes gets chance to taste and reattach.

Spoiling Cartoons

Disk device driver calls disk_create()
and the DISK class creates a new geom.

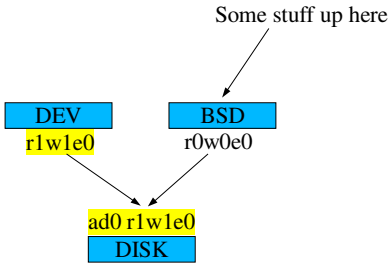
Spoiling Cartoons

NEW_PROVIDER event triggers
a round of tasting. DEV always grabs.
BSD discovers label on disk and grabs.



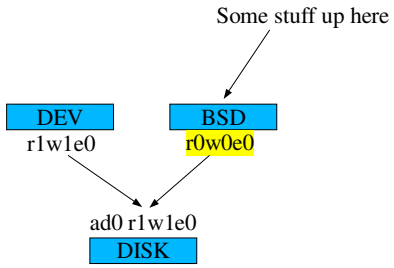
Spoiling Cartoons

We open /dev/ad0 for writing



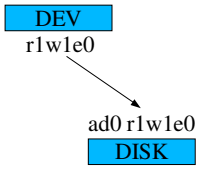
Spoiling Cartoons

write access count goes non-zero
and we spoil the BSD geom.



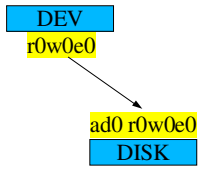
Spoiling Cartoons

BSD geom decides to
self destruct.



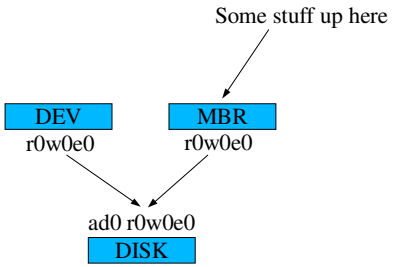
Spoiling Cartoons

The DEV is closed again.



Spoiling Cartoons

A new round of tasting starts
And now MBR finds a label.



This is why...

- You cannot open /dev/ad0 for writing if any slices or labels are open.
- This is policy in the slicer classes, not in GEOM.
- Each geom/class must decide for itself how to react to spoiling.

Management interface(s).

- GEOM needs to be able to report config to userland.
- Since we don't know what the classes are and what they can do, we cannot know what they would like to report.
- => use extensible format.

XML in the KERNEL ???

- No, “XML out of the kernel”.
- There is no point in inventing my own hierarchal extensible modular format when there is one with a lot of tools and growing recognition already.
- Generating XML in the kernel is simple:
  - sbufs - string buffers with memory management.
  - sprintf.

User instruction channel.

- /dev/geom.ctl
  - Implemented in GEOMCTL class.
  - Prefer device over sysctl because it offers access control mechanisms people can understand.
  - Unified command interface.
  - Barely used yet.

Summary:

- GEOM:
  - pushes I/O requests through geoms.
  - posts and runs the events.
- CLASSES:
  - Implement policy.
  - Implement I/O transformations.

“Slicers” as a concept

- “Slicers” are GEOM classes which partition a device into some number of sub devices.
- Commonality includes:
  - Transformation consists of offset + limit.
  - Refuse overlapping slices from opening.
  - On-the-fly change of slice configuration.

Trying to raise the bar...

- Use explicit encode/decode for on-disk meta data.
- This gives the geom modules wordsize/endianess agility.
- Take i386 disk put it in sparc64 and access the partitions.
- Not really that useful until file systems are agile as well.

So what does a slicer take ?

- Four “hard” routines:
  - decode on-disk format into a C-struct.
  - encode C-struct to on-disk format.
  - taste the disk and detect on-disk format.
  - configure slice from C struct.
- Various filler stuff.

Some fine points.

- Remember that there are 3 I/O primitives:
  - Read, Write and Delete.
- Delete is useful in security and for certain storage technologies
  - NAND Flash for instance.

More details

- ioctls gets turned into SETATTR and GETATTR internal GEOM I/O primitives.
- Simplifies just about everything.
- One drawback: copyin/copyout not possible from up/down thread context.
- Fix: “CALL ME” event and EDIRIOCTL pseudo return code.

CALL ME event

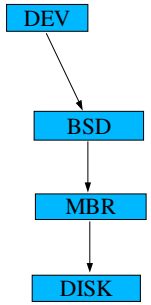
- Says “Please call me from the event queue”.
- Use this for doing things which would sleep in the up/down I/O path.
  - Typically if you need the topology lock.

EDIRIOCTL

- If an ioctl needs copyin/copyout or other similar operations.
- geom's start routine returns bio with pointer to handling function and error = EDIRIOCTL.
- DEV class will call function in users original context where copyin/copyout works as advertised.

WHY ?

ioctl(fd, SOMEFOOIOCTL, bla)



DEV doesn't know which layer wants this ioctl.

Convert ioctl to struct bio, send it down, until somebody says “mine”

EDIRIOCTL gives option of handling in original context.

DISK sends ioctl into device driver, always uses EDIRIOCTL.

On-the-fly changes...

- I would prefer a common on-the-fly change API for all slicers, rather than special ioctls for each CLASS.
- Basic API will be “read special”, “write special” using /dev/geom.ctl interface.
- Concept being:
  - userland prepares what should go on the disk
  - GEOM module validates and reconfigures.

Status:

- Had it almost debugged.
- “Lost” tree in CVS conflicts.
- Working hard to get it reinstated.

And now for something entirely different: Source code...

... lets go look at some code!

What then is GEOM ?

- GEOM is an entirely new way to think about disk-like storage I/O requests.
- GEOM is very very very general compared to what we had before.
  - New possibilities.
  - New problems.
    - What if two providers both want to be “ad0s1” ?

Status of GEOM...

- GEOM does enough now to replace the existing code.
- Minor (temporary) losses of functionality
  - boot0cfg, fdisk, pending “read/write special”.
- Major new functionality
  - sunlabel, gpt, GBDE
- MAJOR new possibilities.

What took you so long ?

- I started on this before 386BSD, on Minix.
- A number of roadblocks killed my prototypes:
  - Lack of kernel concept of “a device” [dev_t]
  - Missing DEVFS
  - Block device aliasing on vnodes.
  - Kernel dump hack.
- It may sound simple, but you'll get wiser...

Future plans, simple:

- Make CCD one/more GEOM class(es)
- Make disk device drivers geoms.
- Use GEOM for ata-RAID applications.
- Management tools.
- Per consumer/provider IO-stats.

Future plans, simple:

- New and wondrous classes:
  - MURPHY: simulate disk errors.
  - OOPS: copy-on-write with commit/abandon.
  - STRIPE, MIRROR, RAID5, CONCAT: obvious.
  - MD5: add integrity checking.
  - ECC: prepare for media errors (archive use).
  - MPATH: mitigate multiple paths to SAN devices.

Future plans, advanced:

- Implement pluggable disk sorting.
  - Per disk choice of disk-sort algorithm.
- Allow people to play with:
  - I/O priorities.
  - Silly seek elimination.
- Lots of interesting issues.
  - We think we have an idea how to do these.

Future plans, really advanced:

- Mapped/Unmapped scatter/gather struct bio.
  - The next **BIG** thing performance wise!
  - Less copying things around.
  - Better (more likely) clustering.
  - Less KVM pressure.
  - Maybe zero-copy user land->device driver.
- Forces/drives/requires buffer cache redesign.

Vinum and RaidFrame ?

- Ideally, I would like to see:
  - Generic GEOM classes for mirror/stripe/raid5.
  - Configuration drivers which reads various on-disk config formats and DTRT.
- I'm not going to do it
  - I'll let whoever is, do what they want.
  - I may bitch if they hack it too badly though :-)

The End.

- A big thanks to:
  - Robert Watson for finding, taming and keeping the paper tiger on its diet.
  - DARPA/SPAWAR for sponsoring this work under contract N66001-01-C-8035 ("CBOSS"), as part of the DARPA CHATS research program.
  - All the giants whose shoulders we stand on.
  - FreeBSD developers and users for putting up with me.