

Block devices and volume management in Linux

Krzysztof Lichota
lichota@mimuw.edu.pl

Linux block devices layer

- Linux block devices layer is pretty flexible and allows for some interesting features:
 - Pluggable I/O schedulers
 - I/O prioritizing (needs support from I/O scheduler)
 - Remapping of disk requests (Device Mapper)
 - RAID
 - Various tricks (multipath, fault injection)
 - I/O tracing (blktrace)

struct bio

- Basic block of I/O submission and completion
- Can represent large contiguous memory regions for I/O but also scattered regions
- Scattered regions can be passed directly to disks capable of scatter/gather
- bios can be split, merged with other requests by various levels of block layer (e.g. split by RAID, merged in disk driver with other disk requests)

struct bio fields

- `bi_sector` – start sector of I/O
- `bi_size` – size of I/O
- `bi_bdev` – device to which I/O is sent
- `bi_flags` – I/O flags
- `bi_rw` – read/write flags and priority
- `bi_io_vec` – memory scatter/gather vector
- `bi_end_io` - function called when I/O is completed
- `bi_destructor` – function called when bio is to be destroyed

struct bio usage

- Allocate bio using `bio_alloc()` or similar function
- Fill in necessary fields (start, device, ...)
- Initialize bio vector
- Fill in end I/O function to be notified when bio completes
- Call `submit_bio()/generic_make_request()`
- Example: `process_read()` in dm-crypt

Other I/O submission functions

- Older interfaces for submitting I/O are supported (but deprecated), they work by wrapping bio
- `submit_bh` – submits I/O for single `buffer_head`
- `ll_rw_block()` - submits multiple `buffer_head` requests

struct request

- Represents request to disk device
- Can contain a list of bios for scatter/gather
- Used by low-level I/O handling routines (I/O scheduling, disk drivers)
- More details in [Documentation/block/request.txt](#)

I/O schedulers

- I/O scheduler decides when and which queued disk requests to send to disk
- Scheduler can be switched during runtime separately for any disk (using `/sys/block/DEVICE/queue/scheduler`)
- I/O scheduler must balance between maximizing overall throughput and not starving requests (especially interactive)
- Currently 4 schedulers are included in vanilla kernel, other are in patches (e.g. dynamic switching between basic schedulers based on system workload)

Standard I/O schedulers

- Noop – submits I/O to device, without any rearranging, just merges subsequent requests. Good for devices without seek penalty (like flash drives).
- CFQ – complete fairness queuing. Divides requests into queues based on priority and takes some number of requests (depending on priority) from each queue in round-robin fashion. Supports I/O prioritization. Good for desktops.

Standard I/O schedulers

- Deadline – assigns to each request a deadline after which it will be put as high priority request to prevent starving. Keeps also requests sorted by position. Takes requests from sorted queue unless deadline is passed for some requests. Good for server workloads.
- Anticipatory – holds requests for some time in hope that requests in nearby sectors arrive. Good in some workloads, bad in others. Used to be default for desktops.

I/O schedulers framework

- I/O scheduler is registered as `struct elevator_type`
- This structure contains functions called by I/O scheduling framework (get next request, merge, etc.) which allow implementing scheduling policy
- Details are complicated, quite good explanation is in [this presentation](#)
- Example: `deadline_dispatch_requests()`
(*dispatch_find_request* label)

I/O request priorities

- Linux kernel is prepared for I/O priorities setting (per process/thread)
- I/O scheduling prioritization must be supported (and taken into account) by I/O scheduler
- Currently only CFQ scheduler supports it
- I/O priority can be set using ionice utility
- There are 3 classes (realtime, best effort, idle) and 8 priority levels for best-effort

I/O request tracing

- Linux kernel contains facility for tracing block requests (blktrace)
- Block requests can be traced on various levels (trace I/O events, trace request queues) and when various events occur (insertion into queue, merging, submission to device, completion, split, requeue, ...)
- Data is passed to userspace tools for processing
- Example in [blktrace-example.txt](#)

Logical volume manager

- Logical volume manager is a part of OS which manages storage devices (block devices)
- LVM uses physical block devices (e.g. disk partitions, iSCSI devices, SAN devices, etc.) to create logical block devices (volumes)
- Volume is like partition – you can put filesystem on it, use it for swap, use as raw database device, etc.
- LVM allows flexible changes of storage configuration

Logical volume manager features

- Adding/removing physical devices to logical volumes to increase/decrease capacity
- Dynamic resizing of volumes
- Moving data between physical devices (for example to unplug one disk)
- Snapshotting (creating frozen state of volume, for example for backup)

Device mapper

- Volume management functions in Linux are implemented using generic Device Mapper framework
- Device mapper provides easy way of “remapping” disk requests in various ways (redirect to other place, hold, etc.)
- Device mapper creates another block device which can be stacked on other block devices

Device mapper (2)

- Device mapper modules:
 - Linux 2.6 logical volume manager (LVM2) (dm-linear, dm-snap)
 - RAID 0 and 1 module (dm-stripe, dm-raid1)
 - Multipath module (dm-mpath) - redirects request using other device in case one disk controller fails
 - Device encryption module (dm-crypt), similar to cryptoloop
 - Request delaying module (dm-delay)
 - Zero-filled (dm-zero)

Device Mapper usage

- Module registers new device type using `dm_register_target()`
- Registered structure `struct target_type` contains:
 - name – name of DM module
 - version – version of module
 - set of function hooks used by DM framework

target_type functions

- Important `struct target_type` functions:
 - `map` – I/O request remapping function, the core of remapping functionality
 - `ctr` – constructor of device, gets parameters as string table
 - `dtr` – destructor of device, called when device is destroyed
 - `end_io` – called when I/O is finished
 - `message` – text command for device
 - `status` – returns device status for DM tools
 - `ioctl` – ioctl call to device

DM map() function

- Gets I/O request before it is submitted and can modify it (change target block device, sector number, etc.), return code decides what to do with request:
 - DM_MAPIO_SUBMITTED - map() function dispatched the request
 - DM_MAPIO_REMAPPED - request should be dispatched to remapped device
 - DM_MAPIO_REQUEUE - request should be resubmitted later
- See: `__map_bio()`, `DM_MAPIO_REMAPPED`, `linear_map()`, `stripe_map()`, `snapshot_map()`

RAID

- Linux contains software RAID implementation (RAID0, RAID1, RAID 4, RAID5, RAID6, RAID10) for performance improvements and high availability
- This RAID implementation does not use Device Mapper but provides its own framework for creating RAID implementations (personalities)

RAID (2)

- Some RAID personalities functionality overlaps with Device Mapper counterparts
 - RAID0 (striping)
 - RAID1 (mirroring)
 - Linear concatenation, but not dynamic
 - Multipath
- Some other features can be implemented using RAID framework (for example “faulty” device)

RAID personality

- RAID driver registers its `struct mdk_personality` using `register_md_personality()`
- Most important fields:
 - `make_request` – function performing I/O request
 - `run` – function called to start RAID array
 - `stop` – function called to stop RAID array
 - `hot_add_disk` – dynamically add disk to array
- Example: `raid1_personality`

EVMS

- EVMS = Enterprise Volume Management System
- Set of tools for unified management of RAID/LVM/disks and filesystems
- Handles many partitioning types, filesystems and volume managers using plugins
- Has some unique features (drive linking)
- Created by IBM, not maintained currently, though some signs of revival appear

Bibliography

- <http://lxr.linux.no/linux+v2.6.23.12/Docum>
- <http://lxr.linux.no/linux+v2.6.23.12/Docum>
- <http://en.wikipedia.org/wiki/CFQ>
- http://en.wikipedia.org/wiki/Deadline_sche
- http://en.wikipedia.org/wiki/Anticipatory_sc
- <http://www.wlug.org.nz/LinuxIoScheduler>
- <http://www.cs.ccu.edu.tw/~lhr89/linux-ker>
- http://www.gelato.org/pdf/apr2006/gelato_
- <http://git.kernel.org/?p=linux/kernel/git/ax>

Bibliography

- <http://sources.redhat.com/dm/>
- <http://lxr.linux.no/linux+v2.6.23.12/Docur>
- <http://christophe.varoqui.free.fr/refbook.ht>
- <http://lwn.net/Articles/124703/>
- <http://www.redhat.com/docs/manuals/csgf>
- <http://lxr.linux.no/linux+v2.6.23.12/drivers>
- <http://evms.sourceforge.net/>
- http://en.wikipedia.org/wiki/Enterprise_Vol