sbull-mq

Learning block subsystem by converting a 10 years old driver do blk-mq

Marcos Paulo de Souza
L3 Support
@omarcossouza
mpdesouza@suse.com

Agenda

Purpose of sbull-mq
Block Layer (Single Queue)
Block Layer (Multi Queue)
sbull
Some block API changes from 2.6.10 – 5.2
sbull-mq
Improving block subsystem documentation
References

$whoami

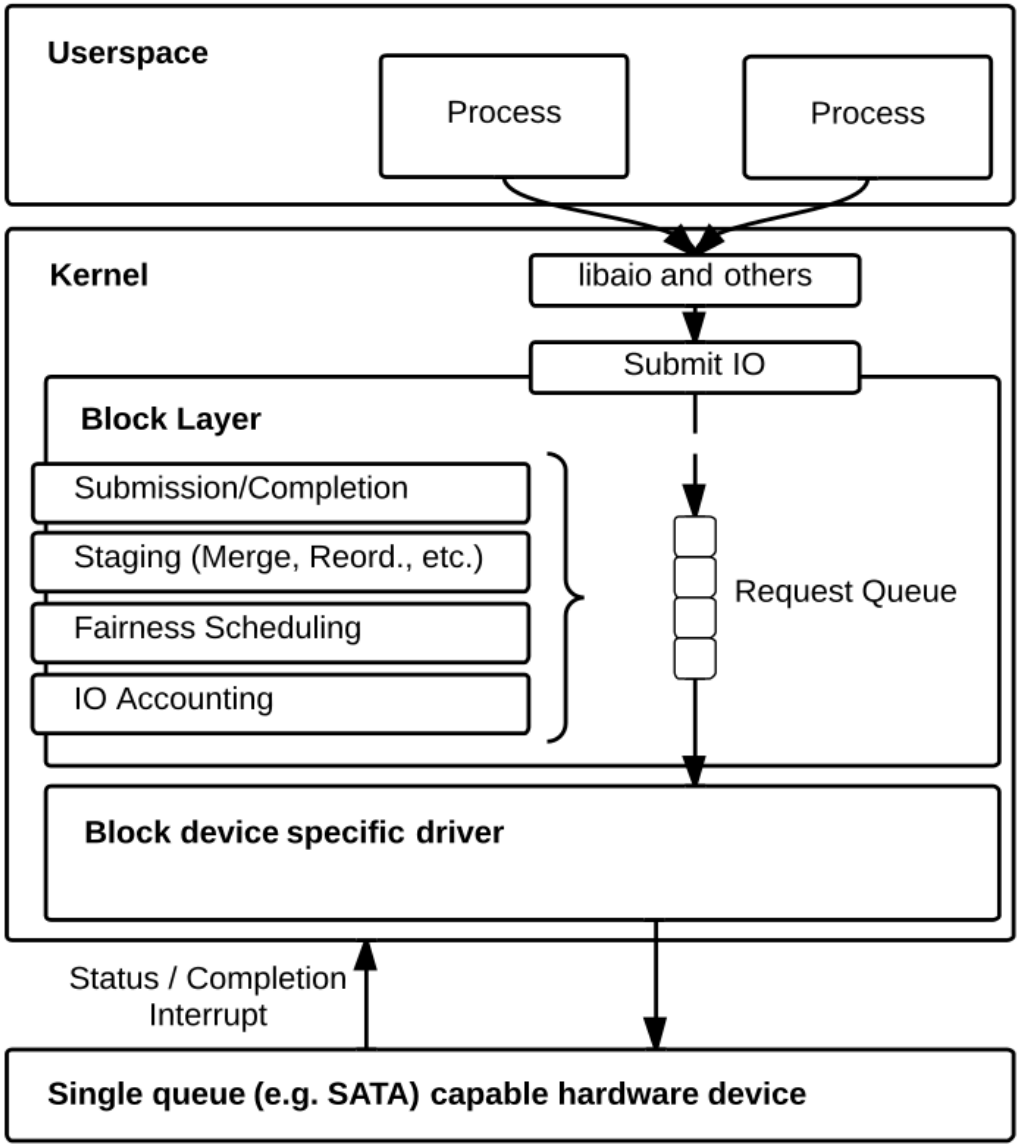
Marcos Paulo de Souza

- L3 Support @ SUSE
- Bachelor in Computer Science at Furb-SC
- Co-founder and organizer of Blumenau HackerSpace (https://hackerspaceblumenau.org)
- Contributor of open source projects like Linux Kernel, LibreOffice, LXC, among others

Purpose of sbull-mq

- Understand the evolution of block subsystem
- Not focused on performance
  - Not now
- Create a fully documented block driver
- Improve block layer by studying the interaction between driver and block API while proposing patches

Block Layer (Single Queue)


Userspace
Process
Process
Kernel
libaio and others
Submit IO
Block Layer
Submission/Completion
Staging (Merge, Reord., etc.)
Fairness Scheduling
IO Accounting
Request Queue
Block device specific driver
Status / Completion
Interrupt
Single queue (e.g. SATA) capable hardware device

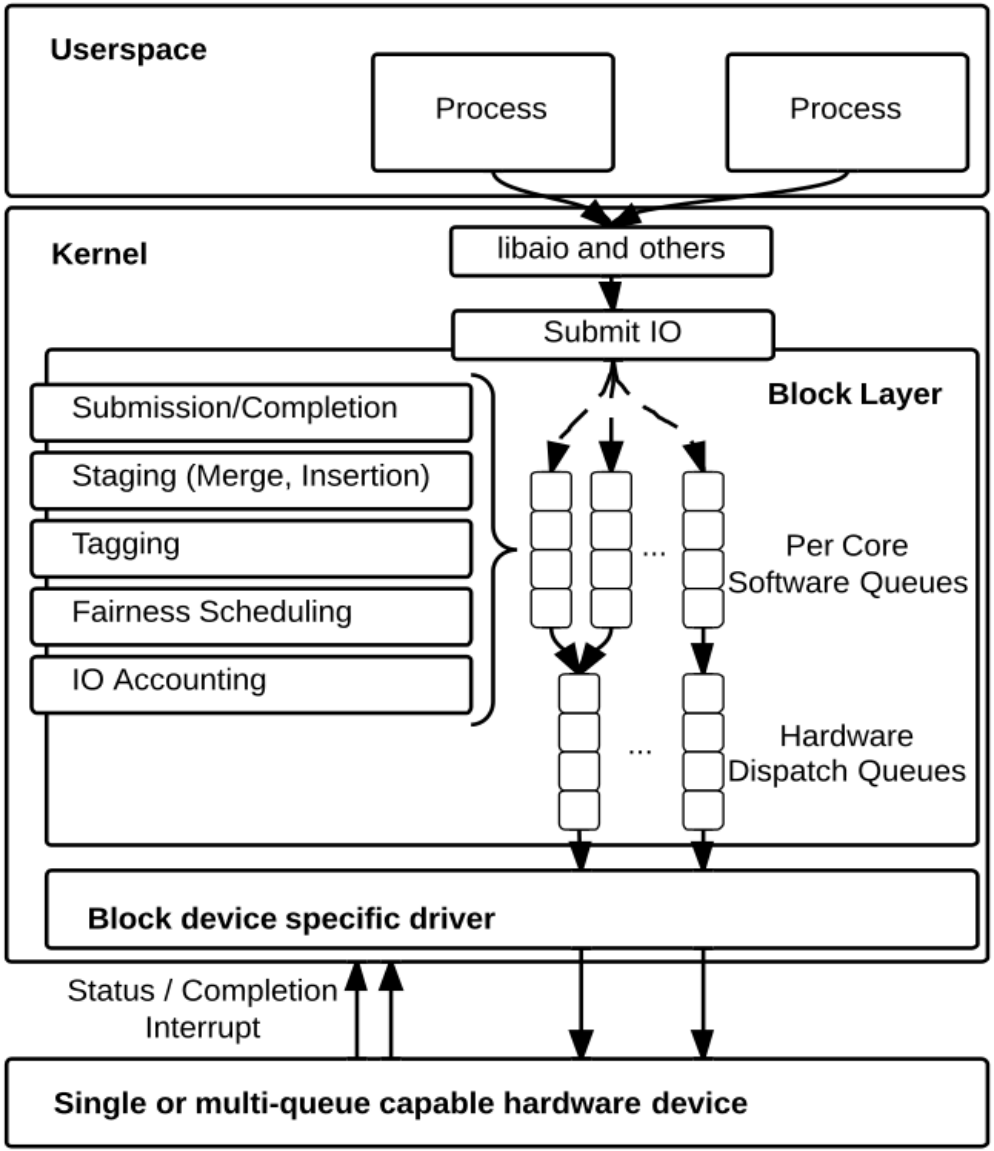
Scalabity problems

- One *Request Queue* per device
  - High lock contention on SMP systems
- Hardware interrupts being delivered to just one core (usually CPU 0)
- Cache Line Invalidation (Remote memory access)
  - Same request queue being shared my multiple cores, cache invalidated

Block Layer (Multi Queue)

- Not something new
  - Merged into kernel 3.13 (2013)

- Aimed to solve the problem mentioned before
  - With some drawbacks:
    - Device drivers should be reworked to enjoy blk-mq
  - With a lot of advantages:
    - SCSI showed up to 8x performance improvement on SQ devices
    - Less code in device drivers


Userspace
Process
Process
Kernel
libaio and others
Submit IO
Block Layer
Submission/Completion
Staging (Merge, Insertion)
Tagging
Fairness Scheduling
IO Accounting
...
Per Core
Software Queues
...
Hardware
Dispatch Queues
Block device specific driver
Status / Completion
Interrupt
Single or multi-queue capable hardware device

Problems solved

- Software Staging Queues
  - A request queue per core/socket
    - Avoids high lock contention by different CPUs
    - Avoids remote cache access and cache invalidation
- Hardware Dispatch Queues
  - One hardware queue per hardware context supported by the device
  - Lockless insertions from staging queue
  - Tagged IO
  - Uses MSI-X, exposes the vector affinity to block layer, avoiding some IPIs

sbull

- Linux Device Drivers 3rd edition's example of a block device driver
  - ramdisk
- Very old
  - It's from 2005, 14 years ago…
- Based in Linux 2.6.10
  - Lot's of things changed since then…

Block API changes from 2.6.10 – 5.2

Lots of functions and structs changed and/or were removed

- elv_next_request
- {blk_}end_request{_all}
- blk_fetch_request
- blk_fs_request
- blk_queue_hardsect_size
- blk_peek_request
- ...

- bio_cur_sectors
- bio_endio
- __bio_k{un}map_atomic
- ...

- request -> {buffer,sector,current_nr_sectors}
- bio -> {bi_size,bi_sector}
- ...

sbull-mq

- https://github.com/marcosps/sbull-mq
- sbull + blk-mq framework = sbull-mq
- Can create a ramdisk with specified amount of memory as storage
  - Supports ext4, btrfs, xfs, maybe others
- How to use
  - insmod ./sbull-mq.ko ndevices=2 disk_size=128M
  - mkfs.btrfs /dev/sbull[ab]
  - mount /dev/sbulla /mnt

- Difficulties to convert to blk-mq:
  - Block API changed a lot
  - Documentation not so clear
  - Archaeological code
  - Misunderstood concept of sectors

- Block API changed a lot and archaeological code:
  - Functions were removed, which one to use instead?
  - Block layer had a different behavior when dealing with requests
    - Kernel 2.6.10 would not remove a request from request_queue if there were BIOs not processed
    - Kernel 5.2+ remove the request once .queue_rq is called, period.

- Some functions are not documented at all
  - Other drivers as example
  - Some magic numbers

- Misunderstood concept of sectors:
  - set_capacity expects size in *SECTORS* of 512 bytes
    - Took me some time to understand it…
    - Driver needs to calculate the capacity by dividing the number of bytes by 512

```
long long sbull_size = memparse("256M", NULL); // returns 268435456, 256Mb conveerted in bytes
sbull_size = sbull_size >> 9; // same as sbull_size / 512, results in 524288 (sectors)
set_capacity(dev->gd, sbull_size); // 524288 sectors of 512 bytes, equivalent to 256Mb
```

Improving block subsystem documentation

- Merged

```
commit 1e9364283764ac93b012739890a30d73e76396db
Author: Marcos Paulo de Souza <marcos.souza.org@gmail.com>
Date:   Sun Feb 10 15:22:51 2019 -0200

    blk-sysfs: Rework documention of __blk_release_queue

    The Notes section of the comment was removed, because now
    blk_release_queue can only be executed from blk_cleanup_queue (being
    called when the q->kobj reaches zero), and also blk_init_queue was removed
    in a1ce35fa4985.

    Signed-off-by: Marcos Paulo de Souza <marcos.souza.org@gmail.com>
    Signed-off-by: Jens Axboe <axboe@kernel.dk>
```

- NAKed

```
From: Marcos Paulo de Souza <marcos.souza.org@gmail.com>
To: linux-block@vger.kernel.org
Cc: Marcos Paulo de Souza <marcos.souza.org@gmail.com>
Subject: [PATCH v2 0/2] Introduce bytes_to_sectors helper in blkdev.h
Date: Wed,  1 May 2019 22:57:26 -0300
Message-ID: <20190502015728.71468-1-marcos.souza.org@gmail.com> (raw)

Changes from v2:
Rename size_to_sectors o bytes_to_sectors. (suggested by Martin K. Petersen)

Changes from v1:
Reworked the documentation of size_to_sectors by removing a sentence that was
explaining the size -> sectors math, which wasn't necessary given the
description prior to the example. (suggested by Chaitanya Kulkarni)

Let me know if you have more suggestions to this code.

Here is the cover letter of the RFC sent prior to this patchset:

While reading code of drivers/block, I was curious about the set_capacity
argument, always shifting the value by 9, and so I took me a while to realize
this is done on purpose: the capacity is the number of sectors of 512 bytes
related to the storage space.

Rather the shifting by 9, there are other places where the value if shifted by
SECTOR_SHIFT, which is more readable.
This patch aims to reduce these differences by adding a new function called
bytes_to_sectors, adding a proper comment explaining why this is needed.

null_blk was changed to use this new function.

Thanks,
Marco

Marcos Paulo de Souza (2):
  blkdev.h: Introduce bytes_to_sectors helper function
  null_blk: Make use of bytes_to_sectors helper

 drivers/block/null_blk_main.c | 18 +++++++++---------
 include/linux/blkdev.h        | 17 +++++++++++++++++
 2 files changed, 26 insertions(+), 9 deletions(-)
```

- Then, a shift problem:

scsi: core: fix the dma_max_mapping_size call

```
We should only call dma_max_mapping_size for devices that have a DMA mask
set, otherwise we can run into a NULL pointer dereference that will crash
the system.

Also we need to do right shift to get the sectors from the size in bytes,
not a left shift.

Fixes: bdd17bdef7d8 ("scsi: core: take the DMA max mapping size into account")
Reported-by: Bart Van Assche <bvanassche@acm.org>
Reported-by: Ming Lei <tom.leiming@gmail.com>
Tested-by: Guilherme G. Piccoli <gpiccoli@canonical.com>
Signed-off-by: Christoph Hellwig <hch@lst.de>
Signed-off-by: Martin K. Petersen <martin.petersen@oracle.com>
```

Diffstat

-rw-r--r- drivers/scsi/scsi_lib.c 6

1 files changed, 4 insertions, 2 deletions

```
diff --git a/drivers/scsi/scsi_lib.c b/drivers/scsi/scsi_lib.c
index 9381171c2fc0..11e64b50497f 100644
--- a/drivers/scsi/scsi_lib.c
+++ b/drivers/scsi/scsi_lib.c
@@ -1784,8 +1784,10 @@ void __scsi_init_queue(struct Scsi_Host *shost, struct request_queue *q)
                blk_queue_max_integrity_segments(q, shost->sg_prot_tablesize);
        }

-       shost->max_sectors = min_t(unsigned int, shost->max_sectors,
-                       dma_max_mapping_size(dev) << SECTOR_SHIFT);
+       if (dev->dma_mask) {
+               shost->max_sectors = min_t(unsigned int, shost->max_sectors,
+                               dma_max_mapping_size(dev) >> SECTOR_SHIFT);
+       }
        blk_queue_max_hw_sectors(q, shost->max_sectors);
        if (shost->unchecked_isa_dma)
                blk_queue_bounce_limit(q, BLK_BOUNCE_ISA);
```

- Merged:

```
commit 327fe1d42b83f8a06b33ba30159582b49af5fc8e
Author: Marcos Paulo de Souza <marcos.souza.org@gmail.com>
Date:   Tue Jul 23 00:27:41 2019 -0300

    block: blk-mq: Remove blk_mq_sched_started_request and started_request

    blk_mq_sched_completed_request is a function that checks if the elevator
    related to the request has started_request implemented, but currently, none of
    the available IO schedulers implement started_request, so remove both.

    Signed-off-by: Marcos Paulo de Souza <marcos.souza.org@gmail.com>
    Signed-off-by: Jens Axboe <axboe@kernel.dk>
```

Questions?

Recerences

- Jens Axboe. blk-mq merge commit. https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=0910c0bdf7c291a41bc21e40a97389c9d4c1960d
- Bjorling, et al. Linux Block IO: Introducing Multi-queue SSD Access on Multi-core Systems

Thanks!

Marcos Paulo de Souza
L3 Support
@omarcossouza
mpdesouza@suse.com


SUSE®
We adapt. You succeed.