ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет безопасности информационных технологий

Дисциплина: «Операционные системы»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

‹‹Планировщик››

Выполнил:
Студент группы N3249
Чан Нгок Хуан

Проверил:
Савков Сергей Витальевич

Санкт-Петербург
2022г.

Лаб 4. Планировщик

1. Провести тестирование и найти лучший планировщик ввода-вывода среди других.

2. Усложнение: Модифицировать существующий планировщик на уровне ядра

Планирование IO

https://habr.com/ru/post/81504/

```
DISC="sda"; \
cat /sys/block/$DISC/queue/scheduler; \
for T in kyber bfq none; do \
        echo $T > /sys/block/$DISC/queue/scheduler; \
        cat /sys/block/$DISC/queue/scheduler; \
        sync && /sbin/hdparm -tT /dev/$DISC && echo "----"; \
        sleep 15; \
done
```

```
Distributor ID: Kali
Description    : Kali GNU/Linux Rolling
Kernel: Linux 5.10.0-kali3-amd64 x86-64
Memory         : 3GB
Processors     : 2
```

1. **Провести тестирование и найти лучший планировщик ввода-вывода среди других**

mq-deadline

DEADLINE is a latency-oriented I/O scheduler. Each I/O request is assigned a deadline. Usually, requests are stored in queues (read and write) sorted by sector numbers. The DEADLINE algorithm maintains two additional queues (read and write) in which requests are sorted by deadline. As long as no request has timed out, the “sector” queue is used. When timeouts occur, requests from the “deadline” queue are served until there are no more expired requests. Generally, the algorithm prefers reads over writes.

MQ-DEADLINE is a latency-oriented I/O scheduler.

kyber

KYBER is a latency-oriented I/O scheduler. It makes it possible to set target latencies for reads and synchronous writes and throttles I/O requests in order to try to meet these target latencies.

Has two request queues:

+ Synchronous requests (e.g. blocked reads)
+ Asynchronous requests (e.g. writes)

bfq

BFQ is a fairness-oriented scheduler. It is described as "a proportional-share storage-I/O scheduling algorithm based on the slice-by-slice service scheme of CFQ. But BFQ assigns budgets, measured in number of sectors, to processes instead of time slices."

BFQ allows to assign I/O priorities to tasks which are taken into account during scheduling decisions.

In addition to cgroups support (blkio or io controllers), BFQ's main features are:

+ BFQ guarantees a high system and application responsiveness, and a low latency for time-sensitive applications, such as audio or video players;
+ BFQ distributes bandwidth, and not just time, among processes or groups (switching back to time distribution when needed to keep throughput high).

none

A trivial scheduler that only passes down the I/O that comes to it. Useful for checking whether complex I/O scheduling decisions of other schedulers are causing I/O performance regressions.

This scheduler is recommended for setups with devices that do I/O scheduling themselves, such as intelligent storage or in multipathing environments. If you choose a more complicated scheduler on the host, the scheduler of the host and the scheduler of the storage device compete with each other. This can decrease performance. The storage device can usually determine best how to schedule I/O.

For similar reasons, this scheduler is also recommended for use within virtual machines.

The NOOP scheduler can be useful for devices that do not depend on mechanical movement, like SSDs. Usually, the DEADLINE I/O scheduler is a better choice for these devices. However, NOOP creates less overhead and thus can on certain workloads increase performance**.**

When NONE is selected as I/O elevator option for blk-mq, no I/O scheduler is used, and I/O requests are passed down to the device without further I/O scheduling interaction.

NONE is the default for NVM Express devices. With no overhead compared to other I/O elevator options, it is considered the fastest way of passing down I/O requests on multiple queues to such devices.

- Проверим планировщики на компьютере:

```
┌──(huan㉿kali)-[~]
└─$ cat /sys/block/sda/queue/scheduler
[mq-deadline] none
```

-Установим планировщик **kyber**:

```
┌──(huan㉿kali)-[~]
└─$ sudo modprobe kyber-iosched
[sudo] password for huan:

┌──(huan㉿kali)-[~]
└─$ cat /sys/block/sda/queue/scheduler
[mq-deadline] kyber none
```

-Установим планировщик **bfq**:

```
┌──(huan㉿kali)-[~]
└─$ sudo modprobe bfq

┌──(huan㉿kali)-[~]
└─$ cat /sys/block/sda/queue/scheduler
[mq-deadline] kyber bfq none
```

- Код:

```
DISC="sda"; \
cat /sys/block/$DISC/queue/scheduler; \
for T in mq-deadline kyber bfq none; do \
        echo $T > /sys/block/$DISC/queue/scheduler; \
        cat /sys/block/$DISC/queue/scheduler; \
        sync && /sbin/hdparm -tT /dev/$DISC && echo "____"; \
        sleep 15; \
done
```

- Результаты:

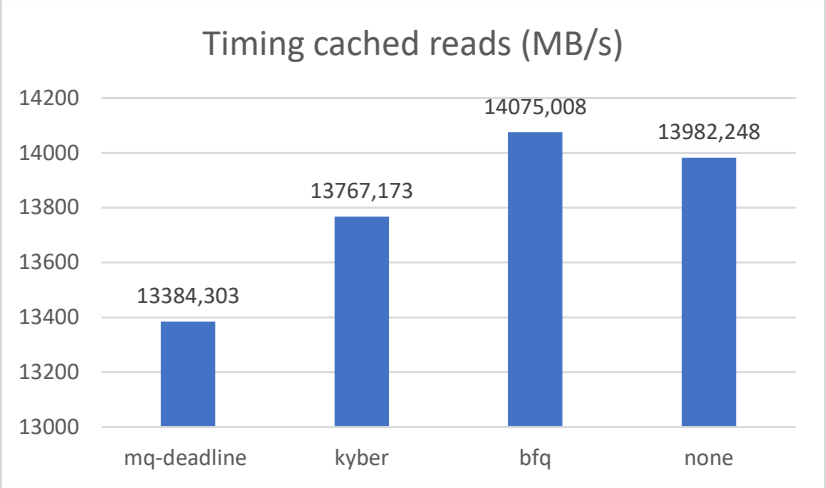
№	mq-deadline		kyber		bfq		none	
	Timing cached reads (MB/s)	Timing buffered disk reads (MB/s)	Timing cached reads (MB/s)	Timing buffered disk reads (MB/s)	Timing cached reads (MB/s)	Timing buffered disk reads (MB/s)	Timing cached reads (MB/s)	Timing buffered disk reads (MB/s)
1	12569,44	884,35	12138,99	896,18	13882,21	855,83	13974,41	884,49
2	13726,21	897,33	13590,31	963,86	13849,13	953,99	13554,52	970,35
3	12828,64	935,84	13371,28	861,86	14298,79	882,73	13498,4	886,66
4	13557,67	917,62	13593,01	887,04	14933,93	921,33	14338,33	923,75
5	13205,58	933,18	13650,04	905,02	13768,07	901,98	13869,38	868,99
6	14133,25	929,87	14018	925,24	14025,09	877,13	14228,8	929,92
7	11190,15	920,11	14258,77	963,86	13539,62	933,99	14068,94	931,85
8	13701,33	955,3	14001,66	906,87	14136,88	914,94	14018,1	914,2
9	14092,19	893,72	14116,05	903,2	14065,56	904,84	14086,29	891,59
10	13422,18	900,18	14227,29	927,52	13884,24	927,35	14105,78	892,65
11	14074,91	908,41	13963,7	934,42	14158,47	871,28	14043,71	901,91
12	14110,09	905,8	14276,98	860,32	14358,11	866,31	14000,31	929,64
average	13384,303	915,143	13767,173	911,283	14075,008	900,975	13982,248	910,500

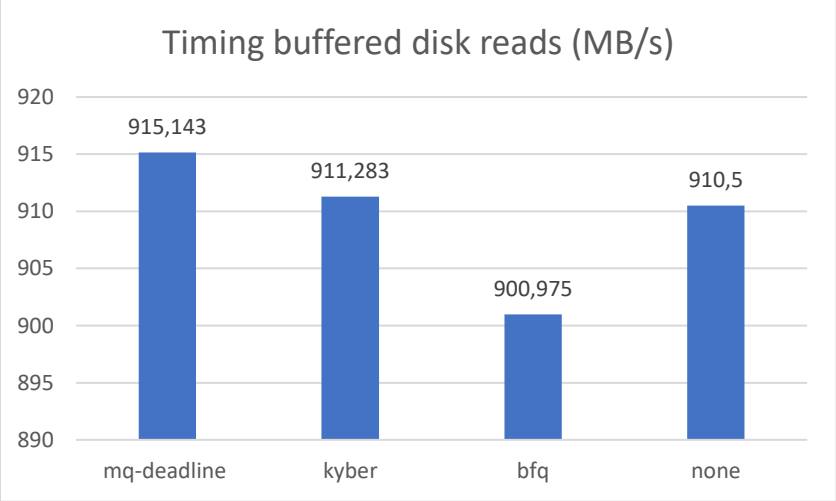
- Запустим скрипт: (первые 2 раза)

```
┌──(huan㉿kali)-[~]
└─$ sudo bash check.sh
[mq-deadline] kyber bfq none
[mq-deadline] kyber bfq none

/dev/sda:
 Timing cached reads:   25108 MB in  2.00 seconds = 12569.44 MB/sec
 Timing buffered disk reads: 2654 MB in  3.00 seconds = 884.35 MB/sec
----
mq-deadline [kyber] bfq none

/dev/sda:
 Timing cached reads:   24250 MB in  2.00 seconds = 12138.99 MB/sec
 Timing buffered disk reads: 2690 MB in  3.00 seconds = 896.18 MB/sec
----
mq-deadline kyber [bfq] none

/dev/sda:
 Timing cached reads:   27730 MB in  2.00 seconds = 13882.21 MB/sec
 Timing buffered disk reads: 2568 MB in  3.00 seconds = 855.83 MB/sec
----
[none] mq-deadline kyber bfq

/dev/sda:
 Timing cached reads:   27914 MB in  2.00 seconds = 13974.41 MB/sec
 Timing buffered disk reads: 2654 MB in  3.00 seconds = 884.49 MB/sec
----
```

```
┌──(huan㉿kali)-[~]
└─$ sudo bash check.sh
[none] mq-deadline kyber bfq
[mq-deadline] kyber bfq none

/dev/sda:
 Timing cached reads:   27418 MB in  2.00 seconds = 13726.21 MB/sec
 Timing buffered disk reads: 2692 MB in  3.00 seconds = 897.33 MB/sec
----
mq-deadline [kyber] bfq none

/dev/sda:
 Timing cached reads:   27148 MB in  2.00 seconds = 13590.31 MB/sec
 Timing buffered disk reads: 2892 MB in  3.00 seconds = 963.86 MB/sec
----
mq-deadline kyber [bfq] none

/dev/sda:
 Timing cached reads:   27664 MB in  2.00 seconds = 13849.13 MB/sec
 Timing buffered disk reads: 2862 MB in  3.00 seconds = 953.99 MB/sec
----
[none] mq-deadline kyber bfq

/dev/sda:
 Timing cached reads:   27076 MB in  2.00 seconds = 13554.52 MB/sec
 Timing buffered disk reads: 2912 MB in  3.00 seconds = 970.35 MB/sec
----
```

Графики:





Вывод: По результатам тестирования:
+ Timing cached reads: mq-deadline < kyber < none < bfq
+ Timing buffered disk reads: bfq < none < kyber < mq-deadline

2. **Усложнение: Модифицировать существующий планировщик на уровне ядра**

BFQ is a fairness-oriented scheduler. It is described as "a proportional-share storage-I/O scheduling algorithm based on the slice-by-slice service scheme of CFQ. But BFQ assigns budgets, measured in number of sectors, to processes instead of time slices."
BFQ allows to assign I/O priorities to tasks which are taken into account during scheduling decisions.
- Установим и модифицируем планировщик ввода-вывода по умолчанию на bfq.

```
┌──(root💀kali)-[~]
└─# echo bfq > /sys/block/sda/queue/scheduler

┌──(root💀kali)-[~]
└─# cat /sys/block/sda/queue/scheduler
mq-deadline kyber [bfq] none
```

```
┌──(huan㉿kali)-[~]
└─$ ls /sys/block/sda/queue/iosched
back_seek_max      fifo_expire_async  low_latency  slice_idle     strict_guarantees
back_seek_penalty  fifo_expire_sync   max_budget   slice_idle_us  timeout_sync

┌──(huan㉿kali)-[~]
└─$ grep -v "is" /sys/block/sda/queue/iosched/*
/sys/block/sda/queue/iosched/back_seek_max:16384
/sys/block/sda/queue/iosched/back_seek_penalty:2
/sys/block/sda/queue/iosched/fifo_expire_async:250
/sys/block/sda/queue/iosched/fifo_expire_sync:125
/sys/block/sda/queue/iosched/low_latency:1
/sys/block/sda/queue/iosched/max_budget:0
/sys/block/sda/queue/iosched/slice_idle:8
/sys/block/sda/queue/iosched/slice_idle_us:8000
/sys/block/sda/queue/iosched/strict_guarantees:0
/sys/block/sda/queue/iosched/timeout_sync:124
```

- В планировщик bfq есть 10 параметров (ниже)

File	Description
`slice_idle`	Value in milliseconds specifies how long to idle, waiting for next request on an empty queue. Default is `8`.
`slice_idle_us`	Same as `slice_idle` but in microseconds. Default is `8000`.
`low_latency`	Enables (1) or disables (0) `BFQ`'s low latency mode. This mode prioritizes certain applications (for example, if interactive) such that they observe lower latency. Default is `1`.
`back_seek_max`	Maximum value (in Kbytes) for backward seeking. Default is `16384`.
`back_seek_penalty`	Used to compute the cost of backward seeking. Default is `2`.
`fifo_expire_async`	Value (in milliseconds) is used to set the timeout of asynchronous requests. Default is `250`.
`fifo_expire_sync`	Value in milliseconds specifies the timeout of synchronous requests. Default is `125`.
`timeout_sync`	Maximum time in milliseconds that a task (queue) is serviced after it has been selected. Default is `124`.
`max_budget`	Limit for number of sectors that are served at maximum within `timeout_sync`. If set to `0` `BFQ` internally calculates a value based on `timeout_sync` and an estimated peak rate. Default is `0` (set to auto-tuning).
`strict_guarantees`	Enables (1) or disables (0) `BFQ` specific queue handling required to give stricter bandwidth sharing guarantees under certain conditions. Default is `0`.

- Я изменил 2 значения:

  low_latency

This parameter is used to enable/disable BFQ's low latency mode. By default, low latency mode is enabled. If enabled, interactive and soft real-time applications are privileged and experience a lower latency, as explained in more detail in the description of how BFQ works.

DISABLE this mode if you need full control on bandwidth distribution. In fact, if it is enabled, then BFQ automatically increases the bandwidth share of privileged applications, as the main means to guarantee a lower latency to them.

In addition, as already highlighted at the beginning of this document, DISABLE this mode if your only goal is to achieve a high throughput. In fact, privileging the I/O of some application over the rest may entail a lower throughput. To achieve the highest-possible throughput on a non-rotational device, setting slice_idle to 0 may be needed too (at the cost of giving up any strong guarantee on fairness and low latency).

slice_idle

This parameter specifies how long BFQ should idle for next I/O request, when certain sync BFQ queues become empty. By default slice_idle is a non-zero value. Idling has a double purpose: boosting throughput and making sure that the desired throughput distribution is respected (see the description of how BFQ works, and, if needed, the papers referred there).

As for throughput, idling can be very helpful on highly seeky media like single spindle SATA/SAS disks where we can cut down on overall number of seeks and see improved throughput.

Setting slice_idle to 0 will remove all the idling on queues and one should see an overall improved throughput on faster storage devices like multiple SATA/SAS disks in hardware RAID configuration, as well as flash-based storage with internal command queueing (and parallelism).

So depending on storage and workload, it might be useful to set slice_idle=0. In general for SATA/SAS disks and software RAID of SATA/SAS disks keeping slice_idle enabled should be useful. For any configurations where there are multiple spindles behind single LUN (Host based hardware RAID controller or for storage arrays), or with flash-based fast storage, setting slice_idle=0 might end up in better throughput and acceptable latencies.

Idling is however necessary to have service guarantees enforced in case of differentiated weights or differentiated I/O-request lengths. To see why, suppose that a given BFQ queue A must get several I/O requests served for each request served for another queue B. Idling ensures that, if A makes a new I/O request slightly after becoming empty, then no request of B is dispatched in the middle, and thus A does not lose the possibility to get more than one request dispatched before the next request of B is dispatched. Note that idling guarantees the desired differentiated treatment of queues only in terms of I/O-request dispatches. To guarantee that the actual service order then corresponds to the dispatch order, the strict_guarantees tunable must be set too.

There is an important flipside for idling: apart from the above cases where it is beneficial also for throughput, idling can severely impact throughput. One important case is random workload. Because of this issue, BFQ tends to avoid idling as much as possible, when it is not beneficial also for throughput (as detailed in Section 2). As a consequence of this behavior, and of further issues described for the strict_guarantees tunable, short-term service guarantees may be occasionally violated. And, in some cases, these guarantees may be more important than guaranteeing maximum throughput. For example, in video playing/streaming, a very low drop rate may be more important than maximum throughput. In these cases, consider setting the strict_guarantees parameter.

2.1 Изменение low_latency:

До изменения (по умолчанию):

```
┌──(huan㉿kali)-[~]
└─$ ls /sys/block/sda/queue/iosched
back_seek_max      fifo_expire_sync  slice_idle         timeout_sync
back_seek_penalty  low_latency       slice_idle_us
fifo_expire_async  max_budget        strict_guarantees

┌──(huan㉿kali)-[~]
└─$ grep -v "is" /sys/block/sda/queue/iosched/*
/sys/block/sda/queue/iosched/back_seek_max:16384
/sys/block/sda/queue/iosched/back_seek_penalty:2
/sys/block/sda/queue/iosched/fifo_expire_async:250
/sys/block/sda/queue/iosched/fifo_expire_sync:125
/sys/block/sda/queue/iosched/low_latency:1
/sys/block/sda/queue/iosched/max_budget:0
/sys/block/sda/queue/iosched/slice_idle:8
/sys/block/sda/queue/iosched/slice_idle_us:8000
/sys/block/sda/queue/iosched/strict_guarantees:0
/sys/block/sda/queue/iosched/timeout_sync:124
```

Изменяем значение параметра **low_latency**:

```
┌──(root💀kali)-[~]
└─# echo 0 > /sys/block/sda/queue/iosched/low_latency

┌──(root💀kali)-[~]
└─# cat /sys/block/sda/queue/iosched/low_latency
0
```

После изменения:

```
┌──(huan㉿kali)-[~]
└─$ grep -v "is" /sys/block/sda/queue/iosched/*
/sys/block/sda/queue/iosched/back_seek_max:16384
/sys/block/sda/queue/iosched/back_seek_penalty:2
/sys/block/sda/queue/iosched/fifo_expire_async:250
/sys/block/sda/queue/iosched/fifo_expire_sync:125
/sys/block/sda/queue/iosched/low_latency:0
/sys/block/sda/queue/iosched/max_budget:0
/sys/block/sda/queue/iosched/slice_idle:8
/sys/block/sda/queue/iosched/slice_idle_us:8000
/sys/block/sda/queue/iosched/strict_guarantees:0
/sys/block/sda/queue/iosched/timeout_sync:124
```

Результаты:

	bfq (low_latency = 1)		bfq (low_latency = 0)	
	cached reads (MB/s)	Buffered disk reads (MB/s)	cached reads (MB/s)	Buffered disk reads (MB/s)
1	14773,04	819,4	13650,27	927,83
2	13636,5	807,05	13637,78	936,1
3	11795,32	928,18	13707,95	948,02
4	13827,21	940,48	13426,42	931,89
5	11876,49	908,79	13218,77	897,08
6	13323,86	955,71	14220,41	928,4
7	13535,7	857,25	14845,55	933,64
8	14407,36	924,56	14384,79	938,33
9	13256,7	925,47	14409,03	937,27
10	14125,45	946,9	13681,36	927,87
11	13157,21	880,99	12478,29	903,83
12	13661,93	891,36	13452,81	908,29
average	13448,06417	898,845	13759,4525	926,5458333

2.2 Изменение slice_idle:

Возвращаем значение параметра **low_latency** к значению по умолчанию:

```
┌──(root💀kali)-[~]
└─# echo 1 > /sys/block/sda/queue/iosched/low_latency

┌──(root💀kali)-[~]
└─# cat /sys/block/sda/queue/iosched/low_latency
1
```

Изменяем значение параметра **slide_idle**:

```
┌──(root💀kali)-[~]
└─# echo 0 > /sys/block/sda/queue/iosched/slice_idle
```

```
┌──(huan㉿kali)-[~]
└─$ grep -v "is" /sys/block/sda/queue/iosched/*
/sys/block/sda/queue/iosched/back_seek_max:16384
/sys/block/sda/queue/iosched/back_seek_penalty:2
/sys/block/sda/queue/iosched/fifo_expire_async:250
/sys/block/sda/queue/iosched/fifo_expire_sync:125
/sys/block/sda/queue/iosched/low_latency:1
/sys/block/sda/queue/iosched/max_budget:0
/sys/block/sda/queue/iosched/slice_idle:0
/sys/block/sda/queue/iosched/slice_idle_us:0
/sys/block/sda/queue/iosched/strict_guarantees:0
/sys/block/sda/queue/iosched/timeout_sync:124
```

Результаты:

	bfq (slice_idle = 8)		bfq (slice_idle = 0)	
	cached reads (MB/s)	Buffered disk reads (MB/s)	cached reads (MB/s)	Buffered disk reads (MB/s)
1	14773,04	819,4	14163,78	963,11
2	13636,5	807,05	13826,06	947,97
3	11795,32	928,18	14169,38	939,26
4	13827,21	940,48	13926,53	913,59
5	11876,49	908,79	13900,35	957,77
6	13323,86	955,71	14081,99	935,86
7	13535,7	857,25	13474,84	956,61
8	14407,36	924,56	13884,19	914,31
9	13256,7	925,47	13731,39	944,61
10	14125,45	946,9	14448,49	876,98
11	13157,21	880,99	13695,57	880,2
12	13661,93	891,36	13989,92	945,07
average	13448,06417	898,85	13941,04	931,28

Вывод:

Я изменил параметры **low_latency** и **slice_idle** в **bfq** и увидел небольшое увеличение значений «**timing cached reads**» и «**timing buffered disk reads**».

Литературы

https://documentation.suse.com/sles/12-SP4/html/SLES-all/cha-tuning-io.html

https://www.kernel.org/doc/Documentation/block/bfq-iosched.txt

https://01.org/linuxgraphics/gfx-docs/drm/block/bfq-iosched.html