**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,   
МЕХАНИКИ И ОПТИКИ»**

**Факультет безопасности информационных технологий**

**Дисциплина:**

**“Операционные системы”**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4**

**Выполнил:**

Студент гр. N3249

Шарифуллин Ильдан Айдарович



**Проверил:**

Савков С.В.

Санкт-Петербург

2022г.

**Задание:**

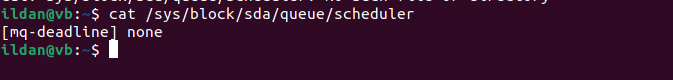
Провести тестирование и найти лучший планировщик ввода-вывода среди других.

Усложнение

Модифицировать существующий планировщик на уровне ядра

**Ход работы:**

По умолчанию в моей системе стоял планировщик mq-deadline



Баш код для перебора и тестирования доступных планировщиков:

#!/bin/bash

DISC="sda"; \

cat /sys/block/$DISC/queue/scheduler; \

for T in kyber bfq none mq-deadline; do \

echo $T > /sys/block/$DISC/queue/scheduler; \

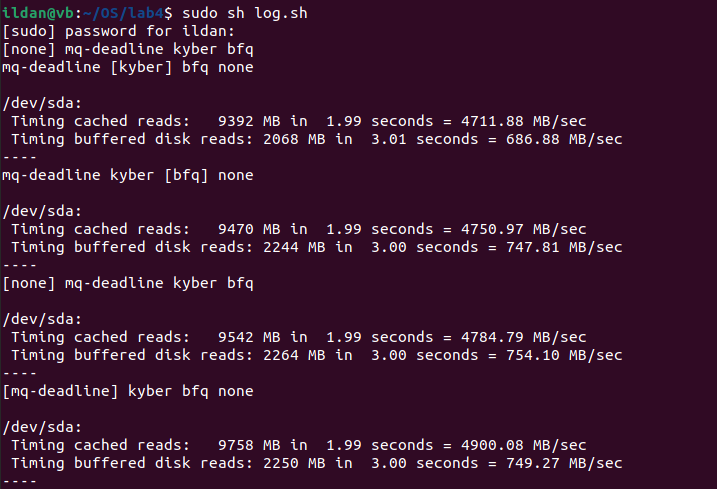
cat /sys/block/$DISC/queue/scheduler; \

sync && /sbin/hdparm -tT /dev/$DISC && echo "----"; \

sleep 15; \

done

Тестирование с использованием приведенного скрипта показало такие результаты:



В скорости кэширования с большим отрывом лидирует mq-deadline, а в буферизации разница между bfq, none, mq-deadline составляет не более 1%. Поэтому mq-deadline определенно является лучшим для моего диска.

Описание планировщиков (взять с ресурса [xakep.ru](https://xakep.ru/)):

* NOOP (none) — наиболее простой планировщик. Он банально помещает все запросы в очередь FIFO и исполняет их вне зависимости от того, пытаются ли приложения читать или писать. Планировщик этот, тем не менее, пытается объединять однотипные запросы для сокращения операций ввода/вывода.
* Deadline в настоящее время является стандартным планировщиком, был разработан в 2002 году. В основе его работы, как это ясно из названия, лежит предельный срок выполнения — то есть планировщик пытается выполнить запрос в указанное время. В дополнение к обычной отсортированной очереди, которая появилась еще в Linus Elevator, в нем есть еще две очереди — на чтение и на запись. Чтение опять же более приоритетно, чем запись. Кроме того, запросы объединяются в пакеты. Пакетом называется последовательность запросов на чтение либо на запись, которая идет в сторону больших секторов («алгоритм лифта»). После его обработки планировщик смотрит, есть ли запросы на запись, которые не обслуживались

длительное время, и в зависимости от этого решает, создавать ли пакет на чтение либо же на запись.

* BFQ (Budget Fair Queueing) — относительно новый планировщик. Базируется на CFQ. Если не вдаваться в технические подробности, каждой очереди (которая, как и в CFQ, назначается попроцессно) выделяется свой «бюджет», и, если процесс интенсивно работает с диском, данный «бюджет» увеличивается.
* Kyber был написан для работы с быстрыми устройствами. Используя две очереди — запросы на запись и на чтение, kyber отдает приоритет запросам на чтение, перед запросами на запись. Алгоритм измеряет время завершения каждого запроса и корректирует фактический размер очереди для достижения установленных в настройках задержек.

Изменение параметр я тестировал на планировщике mq-deadline.

Для увеличения буферизации я решил уменьшить длину дисковой очереди. При этом скорость кэширования уменьшилась. Для коротких промежутков буферизации меньшая длина очереди более оптимальна.

По умолчанию параметр равен:



И действительно, уменьшив длину очереди скорость повысилась. Оптимальный параметр: 32. Ниже привожу результаты тестов:

|  |  |  |
| --- | --- | --- |
|  | 8 | |
| 1 | 4499 | 743 |
| 2 | 4800 | 624 |
| 3 | 4578 | 651 |
| 4 | 4651 | 801 |
| 5 | 4574 | 750 |
| 6 | 4241 | 779 |
| Ṡ | 4557,166667 | 724,6666667 |
| σ | 169,2310222 | 64,95810616 |
| Ṡ + 3σ | 5064,859733 | 919,5409851 |
| Ṡ - 3σ | 4049,4736 | 529,7923482 |

|  |  |  |
| --- | --- | --- |
|  | 16 | |
| 1 | 4425 | 725 |
| 2 | 4774 | 757 |
| 3 | 4571 | 740 |
| 4 | 4610 | 771 |
| 5 | 4493 | 744 |
| 6 | 4537 | 772 |
| Ṡ | 4568,333333 | 751,5 |
| σ | 108,9673753 | 16,93861466 |
| Ṡ + 3σ | 4895,235459 | 802,315844 |
| Ṡ - 3σ | 4241,431207 | 700,684156 |

|  |  |  |
| --- | --- | --- |
|  | 32 | |
| 1 | 4847 | 756 |
| 2 | 4546 | 782 |
| 3 | 4529 | 738 |
| 4 | 4615 | 740 |
| 5 | 4454 | 764 |
| 6 | 4421 | 721 |
| Ṡ | 4568,666667 | 750,1666667 |
| σ | 139,4377599 | 19,7519338 |
| Ṡ + 3σ | 4986,979946 | 809,4224681 |
| Ṡ - 3σ | 4150,353387 | 690,9108653 |

|  |  |  |
| --- | --- | --- |
|  | 64 | |
| 1 | 4484 | 750 |
| 2 | 4425 | 749 |
| 3 | 4436 | 723 |
| 4 | 4473 | 772 |
| 5 | 4511 | 734 |
| 6 | 4348 | 724 |
| Ṡ | 4446,166667 | 742 |
| σ | 52,50846493 | 17,13670525 |
| Ṡ + 3σ | 4603,692061 | 793,4101157 |
| Ṡ - 3σ | 4288,641272 | 690,5898843 |