Previous Chapter - Table of Contents - Next Chapter

Chapter 15 I/O Scheduling - Notes

15.2 Introduction

System performance often depends very heavily on optimizing I/O scheduling strategy. Many (often competing) factors influence behavior, including minimizing hardware access times, avoiding wear/tear on storage media, ensuring data integrity, granting timely access to applications that need to do I/O, being able to prioritize important tasks. Linux offers variety of **I/O Schedulers** to choose from, each of which has tunable parameters + number of utilities for reporting on/analyzing I/O performance.

15.3 Learning Objectives:

- Explain the importance of I/O scheduling and describe the conflicting requirements that need to be satisfied.
- Delineate and contrast the options available under Linux.
- Understand how the **CFQ** (**C**ompletely **F**air **Q**ueue) and **Deadline** algorithms work.

15.4 I/O Scheduling

I/O scheduler provides interface between generic block layer and low-level physical device drivers. Both VM (Virtual Memory) and VFS (Virtual File System) layers submit I/O requests to block devices. Job of I/O scheduling layer to prioritize + order requests before they are given to block devices.

Any I/O scheduling algorithm has to satisfy certain (sometimes conflicting) requirements:

- Hardware access times should be minimized; ie, requests should be order according to physical location on disk. Leads to **elevator** scheme where requests inserted in pending queue in physical order
- Requests should be merged to extent possible to get as big a contiguous region as possible, which also minimizes disk access time
- Requests should be satisfied with as low a latency as feasible. In some cases, determinism (in sense of deadlines) important
- Write operations usually wait to migrate from caches to disk without stalling process. Read operations almost always require process to wait for completion before proceeding further. Favoring reads over writes leads to better parallelism and system responsiveness
- Processes should share I/O bandwidth in fair, consciously prioritized fashion. Even if it means some overall performance slowdown of I/O layer, process throughput should not suffer inordinately

15.5 I/O Scheduler Choices

Since demands can be conflicting, different I/O schedulers may be appropriate for different workloads, eg, large database server vs. desktop system. Different hardware may mandate different strategy. To provide flexibility, Linux kernel has object oriented scheme, where pointers to various needed functions supplied in a data structure, the particular one of which can be selected at boot on kernel command line:

```
linux ... elevator=[cfq|deadline|noop]
```

At least one of I/O scheduling algorithms must be compiled into kernel. Current choices:

- Completely Fair Queueing (CFQ)
- Deadline Scheduling

- noop (A simple scheme)

Default choice is compile configuration option. Modern distributions choose either CFQ or Deadline.

15.6 I/O Scheduling and SSD Devices

Gradual introduction of **SSD** (**S**olid **S**tate **D**rive) devices, which use flash memory to emulate hard disks, has important implications for I/O scheduling.

SSDs do not require elevator scheme + benefit from **wear leveling** to spread I/O over the devices which have limited write/erase cycles.

Can examine `/sys/block/<device>/queue/rotational` to see whether or not device is **SSD** or not:

```
$ cat /sys/block/sda/queue/rotational
1
$ cat /sys/block/sdb/queue/rotational
0
```

15.7 Tunables and Switching the I/O Scheduler at Runtime

Each of I/O schedulers exposes parameters which can be used to tune behavior at run time. Parameters accessed through pseudo-filesystem mounted at `/sys`.

In addition, possible to use different I/O schedulers for different devices. Choice can be made easily through command line. For example:

```
$ cat /sys/block/sda/queue/scheduler
noop deadline [cfq]
$ echo noop > /sys/block/sda/queue/scheduler
$ cat /sys/block/sda/queue/scheduler
[noop] deadline cfq
```

Actual tunables vary according to particular I/O scheduler, can be found under `/sys/block/<device>/queue/iosched`.

Can see actual tunables for disk using **CFQ** below.

Will discuss some parameters shortly.

cfqio

15.9 CFQ (Completely Fair Queue) Scheduler

CFQ (**C**ompletely **F**air **Q**ueue) method has goal of equal spreading of I/O bandwidth among all processes submitting requests.

Theoretically, each process has own I/O queue, which works together with dispatch queue which receives actual requests on way to device. In actual practice, number of queues fixed (at 64) and hash process based on process ID used to select queue when request submitted.

Dequeuing of requests done **round robin** style on all queues, each one of which works in **FIFO** (**F**irst **I**n **F**irst **O**ut) order. Thus, work spread out. To avoid excessive seeking operations, entire round selected, and then sorted into dispatch queue before actual I/O requests issued to device.

15.10 CFQ Tunables

In examples below, parameter `HZ` -> kernel-configured quantity, corresponds to number of **jiffies** per second, which kernel uses as coarse measure of time. Time unit `HZ/2` = 0.5 seconds, `5 * HZ` = 5 seconds, etc.

- `quantum` : Maximum queue length in one round of service (Default = 4)
- `queued` : Minimum request allocation per queue (Default = 8)
- `fifo_expire_sync` : `FIFO` timeout for sync requests (Default = `HZ/2`)
- `fifo_expire_async` : `FIFO` timeout for async requests (Default = `5 * HZ`)
- `fifo_batch_expire` : Rate at which the `FIFO` 's expire (Default = `HZ/8`)
- `back_seek_max` : Maximum backwards seek, in KB (Default = 16K)
- `back_seek_penalty` : Penalty for a backwards seek (Default = 2)

15.11 Deadline Scheduler

Deadline `I/O` scheduler aggressively reorders requests with simultaneous goals of improving overall performance + preventing large latencies for individual requests, ie, limiting starvation.

Kernel associates **deadline** with each and every request. Read requests = higher priority then write requests.

Five separate `I/O` queues maintained:

- Two sorted lists maintained, one for reading, one for writing, arranged by starting block
- Two **FIFO** lists maintained, one for reading, one for writing. Lists sorted by submission time
- Fifth queue contains requests to be shoveled to device driver itself. Called dispatch queue

Art of the algorithm: how requests are peeled off from first four queues and placed on fifth (dispatch queue).

15.12 Deadline Tunables

Available tunables for **Deadline** scheduler:

- `read_expire` : How long (in milliseconds) read request is guaranteed to occur within (Default = `HZ/2 = 500`)
- `write_expire` : How long (in milliseconds) write request is guaranteed to occur within (Default = `5 * HZ = 5000`)
- `writes_starved` : How many requests we should give preference to reads over writes (Default = `2`)
- `fifo_batch` : How many requests should be moved from sorted scheduler list to dispatch queue, when deadlines have expired (Default = `16`)
- `front_merges` : Back merges more common than front merges as contiguous request usually continues to next block. Setting this parameter to 0 disables front merges and can give boost if you know they are unlikely to be needed (Default = `1`)

##

Back to top

Previous Chapter - Table of Contents - Next Chapter