1. I / O Scheduler 1
   1. BFQ**1**
   2. NONE**3**
   3. Kyber4
   4. MQ-DEADLINE5
2. I / O Scheduler 성능6

2.1 BFQ의 성능6

2.1.1 BFQ의 매개변수8

1. **I / O Scheduler**

Linux에서 실행되는 거의 모든 응용 프로그램은 일종의 I / O를 수행합니다. 웹 서핑조차도 많은 작은 파일들을 디스크에 씁니다. I / O 스케줄러가 없으면 모든 I / O 요청이 커널에 인터럽트를 보내서 I / O 작업을 수행하여 디스크 헤드가 다른 블록 주위로 이동하며 읽기 및 쓰기 요청을 합니다. 시간이 지남에 따라 디스크 드라이브의 성능과 나머지 시스템 간의 성능 차이가 발생하여 전반적인 시스템 성능 안 좋은 영향을 끼치기 때문에 I / O 스케줄러는 매우 중요합니다.

I / O 스케줄러는 디스크 액세스 요청을 최적화하는 방법으로 존재합니다. 이들은 전통적으로 I / O 요청을 디스크의 비슷한 위치에 병합하여 이를 수행합니다. 디스크의 유사한 섹션에있는 요청을 그룹화하면 드라이브가 자주 검색할 필요가 없어지므로 디스크 작업에 대한 전체 응답 시간이 향상됩니다.

I / O 스케줄러는 일반적으로 다음 기술을 사용합니다.

* **병합** 인접한 요청이 병합되어 디스크 찾기를 줄이고 I/O syscalls의 크기를 늘립니다.
* **우선 순위 지정** 요청은 어떤 방식으로든 우선 순위가 결정됩니다. 세부 사항은 I / O 스케줄러에 달려 있습니다.
* **정렬** 디스크상의 물리적 위치를 기준으로 요청이 정렬되어 검색이 가능한 한 방향으로 정렬되어집니다.

Linux 에서는 싱글 큐 스케줄러 **CFQ, NOOP, DEADLINE** 을 사용했었지만 Linux 5.0 버전 이후 커널에서 제거됐고, 현재는 멀티 큐 스케줄러 **BFQ, Kyber, MQ-DEADLINE, NONE** 을 사용하고 있습니다.

**1.1 BFQ (Budget Fair Queueing)**

BFQ는 공정성 지향의 I / O 스케줄러입니다. BFQ는 오디오 및 비디오 플레이어와 같이 시간에 민감한 응용 프로그램에 대해 높은 시스템 및 응용 프로그램 응답성과 낮은 대기 시간을 보장합니다. BFQ는 프로세스 또는 그룹간에 시간뿐만 아니라 대역폭을 분배합니다 (처리량을 높이기 위해 필요할 때 시간 분배로 다시 전환) BFQ의 주요 기능은 다음과 같습니다.

* **대화식 작업을 위한 낮은 대기 시간**

BFQ는 다른 I / O 스케줄러보다 최대 20 배 빠르게 애플리케이션을 로드합니다. 시스템은 백그라운드 I / O 워크로드에 관계없이 유휴 상태 인 것처럼 반응이 빠릅니다.

* **부드러운 실시간 애플리케이션을 위한 낮은 대기 시간**

오디오 및 비디오 플레이어 또는 오디오 오디오 및 비디오 스트리밍 응용 프로그램과 같은 실시간 응용 프로그램은 백그라운드 I / O 작업 부하에 관계없이 부드러운 재생이 가능합니다.

* **강력한 대역폭 및 대기 시간 보장**

다중 클라이언트 응용 프로그램에서 (예 : 여러 클라이언트, 그룹, 컨테이너, 가상 시스템 또는 다른 유형의 엔티티가 공유 매체와 경쟁 할 때) BFQ는 각 엔티티에 강력한 대역폭 및 대기 시간을 보장합니다.

* **멀티 클라이언트 애플리케이션에서 5 배 ~ 10 배 처리량 향상**

BFQ는 스토리지를 경쟁하는 각 엔티티의 대역폭을 보장하기 위해 다른 솔루션보다 5 배에서 10 배 높은 처리량에 도달합니다.

* **코드 개발 작업을위한 더 빠른 속도**

일부 추가 워크로드가 병렬로 실행되는 경우 BFQ는 일반적인 코드 개발 작업 (컴파일, 체크 아웃, 병합 등)의 I / O 관련 구성 요소를 다른 I / O 스케줄러보다 훨씬 빠르게 실행합니다.

특이하게 BFQ는 처리량보다 대기 시간을 단축합니다. 따라서 BFQ는 대기 시간을 단축하기 위해 처리량을 낮추는 일정을 만듭니다.

BFQ 스케줄러는 아래의 매개 변수를 사용할 수 있습니다.

[표 1 : BFQ의 조정 가능한 매개 변수]

|  |  |
| --- | --- |
| **slice\_idle** | 밀리 초 단위의 값을 빈 대기열에서 다음 요청을 기다리는 유휴 시간으로 지정합니다.  기본값은 8입니다. |
| **slice\_idle\_us** | 마이크로 초 단위의 값을 빈 대기열에서 다음 요청을 기다리는 유휴 시간으로 지정합니다.  기본값은 8000입니다. |
| **low\_latency** | 낮은 대기 시간 모드를 활성화 (1) 또는 비활성화 (0) 합니다. 이 모드는 특정 응용 프로그램 (예 : 대화 형인 경우)의 우선 순위를 지정하여 대기 시간이 줄어 듭니다.  기본값은 1입니다. |
| **back\_seek\_max** | 역방향 탐색을 위한 최대 값 (KB)입니다.  기본값은 16384입니다. |
| **back\_seek\_penalty** | 역방향 탐색 비용을 계산하는 데 사용됩니다.  기본값은 2입니다. |
| **fifo\_expire\_async** | 비동기 요청의 시간 초과를 설정하는 데 값 (ms)이 사용됩니다.  기본값은 250입니다. |
| **fifo\_expire\_sync** | 밀리 초 단위의 값은 동기 요청의 시간 초과를 지정합니다.  기본값은 125입니다. |
| **timeout\_sync** | 작업 (큐)이 선택된 후 서비스되는 최대 시간 (ms)입니다.  기본값은 124입니다. |
| **max\_budget** | 최대 범위 내에서 제공되는 섹터 수에 대한 제한 timeout\_sync. 0 BFQ 내부적으로 설정된 경우 timeout\_sync 추정 피크 속도를 기준으로 값을 계산합니다 .  기본값은 0 (자동 조정으로 설정)입니다. |
| **strict\_guarantees** | 특정 조건에서 보다 엄격한 대역폭 공유를 보장하는 데 필요한 특정 대기열 처리를 활성화 (1) 또는 비활성화 (0) 합니다.  기본값은 0입니다. |

**1.2 NONE**

NONE은 스케줄러가 아닙니다. NONE를 엘리베이터 옵션으로 선택하면 I / O 스케줄러가 사용되지 않으며, 대기열 알고리즘이 적용되지 않고 I / O 요청은 추가 I / O 스케줄링 상호 작용 없이 디바이스로 전달됩니다.

NONE은 다른 I/O 엘리베이터 옵션과 달리 오버헤드가 없기 때문에 여러 대기열의 I/O 요청을 해당 장치에 전달하는 가장 빠른 방법입니다.

**1.3 Kyber**

Kyber I / O 스케줄러는 멀티 큐 및 기타 고속 장치에 적합한 오버 헤드가 낮은 지연 시간 중심의 스케줄러이고, 네트워크 라우팅에 사용되는 활성 큐 관리 기술에서 영감을 얻은 스케줄러입니다. 요청을 제한하기 위한 메커니즘으로 사용되는 "토큰"을 기반으로 합니다. 요청을 할당하기 위해 큐잉 토큰이 필요하고, 이는 요청의 기아 상태를 막기 위해 사용됩니다. 또한 디스패치 토큰도 필요하며 주어진 장치에서 특정 우선 순위의 작업을 제한합니다. 마지막으로 대상 읽기 대기 시간이 정의되고 스케줄러가 이 대기 시간 목표에 도달하도록 대기열의 깊이를 자체적으로 조정합니다.

Kyber I / O 스케줄러를 통과하는 I / O 요청은 동기식 요청과 비동기식 요청 또는 읽기와 쓰기를 위한 두 개의 기본 대기열로 나뉘고, 두 대기열을 사용하여 쓰기 요청보다 읽기 요청에 우선 순위를 부여합니다(시작 프로세스는 일반적으로 쓰기가 실제로 발생하는 시점을 신경 쓰지 않습니다. 따라서 쓰기보다 읽기를 우선 순위로 지정하는 것이 일반적입니다). 읽기 요청을 하는 프로세스는 일반적으로 요청이 완료되고 데이터를 사용할 수 있을 때까지 진행할 수 없으므로 이러한 요청은 동기식으로 표현되고 쓰기 작업은 나중에 완료 됩니다.

Kyber I / O 스케줄러는 대기 시간 지향의 스케줄러이기 때문에 목표 대기 시간을 충족시키기 위해 읽기 및 쓰기에 대한 목표 대기 시간을 설정하고 I / O 요청을 제한합니다. 읽기 작업의 대상 대기 시간 기본값은 2000000(ns), 쓰기 작업의 대상 대기 시간 기본값은 10000000(ns)입니다.

[표 2 : Kyber의 조정 가능한 매개 변수]

|  |  |
| --- | --- |
| **read\_lat\_nsec** | 읽기 작업의 대상 대기 시간을 나노초로 설정합니다.  기본값은 2000000입니다. |
| **write\_lat\_nsec** | 쓰기 작업의 대상 대기 시간을 나노초로 설정합니다.  기본값은 10000000입니다. |

Kyber 알고리즘의 핵심은 디스패치 대기열로 전송되는 작업(읽기 및 쓰기)수가 엄격하게 제한되어 대기열이 상대적으로 짧다는 것입니다. 디스패치 큐가 짧은 경우, 주어진 요청이 큐에서 대기하는 동안 경과되는 시간(요청 당 대기 시간)은 상대적으로 작습니다. 이를 통해 우선 순위가 높은 요청에 대한 빠른 처리를 보장합니다.각 요청의 완료 시간을 측정하고 설정에 지정된 지연을 달성하기 위해 실제 큐 크기를 조정합니다. 알고리즘의 구현은 비교적 간단하며 빠른 장치에서 효율적으로 동작합니다.

**1.4 MQ-DEADLINE**

단일 큐 스케줄러는 Linux 5.0 버전 이후로 커널에서 제거되었기 때문에 단일 큐 스케줄러 DEADLINE을 멀티 큐에 적용하여 만든 스케줄러 입니다. 때문에 DEADLINE이 사용하던 매개변수도 그대로 사용합니다.

[표 3 : MQ-DEADLINE과 DEADLINE의 조정 가능한 매개 변수]

|  |  |
| --- | --- |
| **writes\_starved** | 쓰기보다 읽기가 선호되는 횟수를 제어합니다. 쓰기 및 읽기가 동일한 선택 기준으로 디스패치 되기 전에 3 개의 읽기 조작이 수행 될 수 있음 을 의미합니다.  기본값은 3입니다. |
| **read\_expire** | 읽기 작업 의 최종 기한 (현재 시간 + 값)을 밀리 초 단위로 설정합니다.  기본값은 500입니다. |
| **write\_expire** | 쓰기 작업 의 최종 기한 (현재 시간 + 값)을 밀리 초 단위로 설정합니다.  기본값은 5000입니다. |
| **front\_merges** | 전면 병합 요청 시도를 활성화 (1) 또는 비활성화 (0)합니다.  기본값은 1입니다. |
| **fifo\_batch** | 일괄 처리 당 최대 요청 수를 설정합니다 (마감 기한은 일괄 처리에만 검사 됨). 이 매개 변수를 사용하면 대기 시간과 처리량 사이의 균형을 유지할 수 있습니다.  기본값은 16입니다. |

MQ-DEADLINE는 대기 시간 지향적 I / O 스케줄러이고, 요청이 스케줄러에 도달 한 시점부터 요청에 대해 보장 된 대기 시간을 제공합니다. 또한 대기 중인 I/O 요청을 읽기 또는 쓰기 배치로 정렬하고 LBA(논리적 블록 주소 지정) 순서의 실행을 예약합니다.

응용 프로그램이 읽기 I/O 작업을 차단할 가능성이 높기 때문에 기본적으로 읽기 배치는 쓰기 배치보다 우선합니다. MQ-DEADLINE 배치를 처리 한 후 쓰기 작업에 필요한 프로세서 시간을 확인하고 다음 읽기 또는 쓰기 배치를 적절하게 예약합니다.

이 스케줄러는 대부분의 사용 사례에 적합하지만 특히 읽기 작업이 쓰기 작업보다 자주 발생하는 경우에 더욱 적합하고, 상당히 낮은 CPU 오버 헤드로 우수한 성능을 제공합니다.

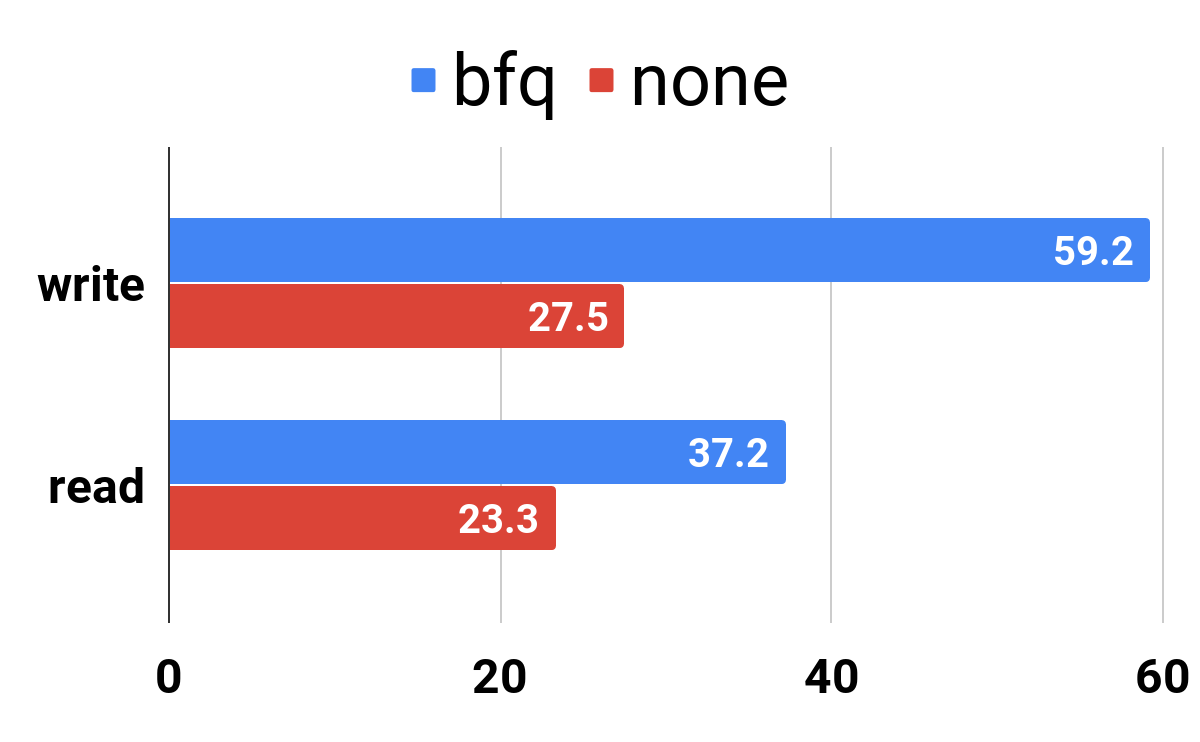
1. **I / O scheduler 성능**

**2.1 bfq의 성능**

bfq의 성능을 체감하기 위해서 큐잉 알고리즘이 적용되지 않은 none과 성능 비교를 위해 fio 명령을 이용해서 한 번에 4개의 프로세스를 실행하는 파일을 none과 bfq의 성능을 비교했습니다.

fio --name=randwrite --ioengine=libaio --iodepth=1 --rw=write --bs=4k --direct=1 --size=512M --runtime=60 --group\_reporting --numjobs=4

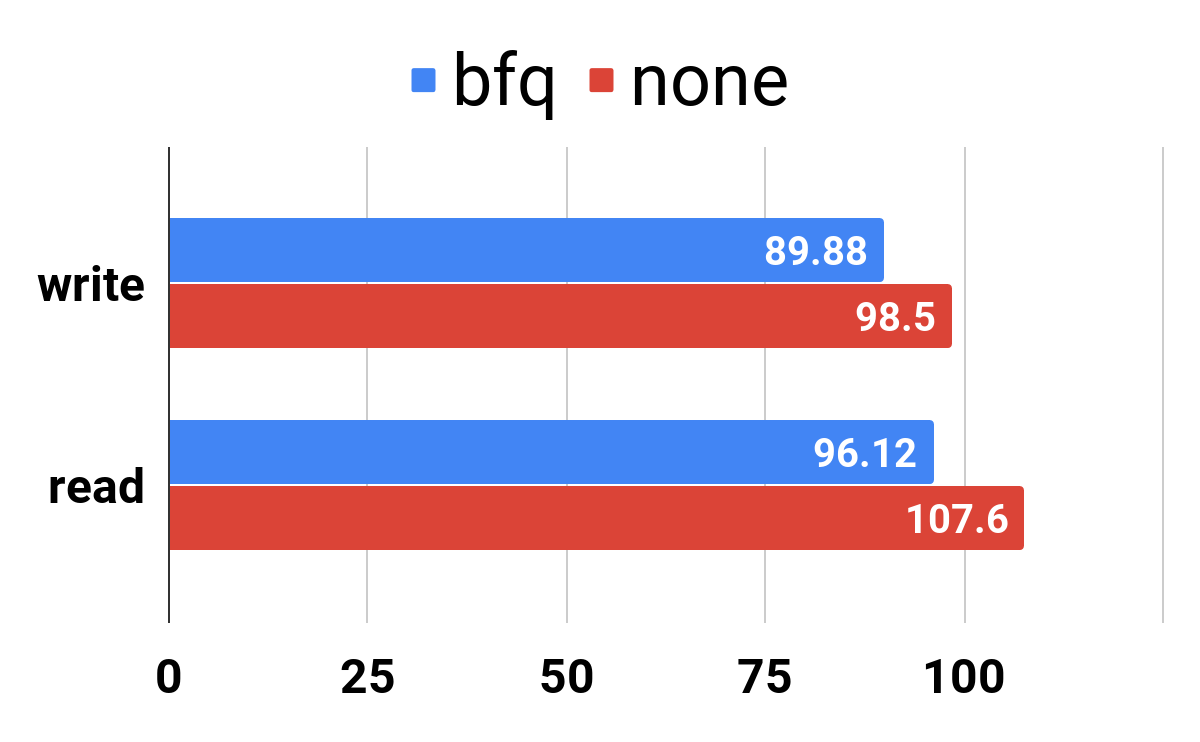
[차트1 bfq와 none 한 번에 4개의 프로세스 실행]



위의 차트1을 보면 한 번에 4개의 프로세스를 실행할 때, 큐잉 알고리즘이 적용되지 않은 none와 비교하면 bfq 스케줄러는 write를 할 때 none 보다 2배 이상 빨랐고, read를 할 때 약 50% 이상 빨랐습니다.

fio --name=randwrite --ioengine=libaio --iodepth=1 --rw=write --bs=4k --direct=1 --size=512M --runtime=60 --group\_reporting --numjobs=1

[차트2 bfq와 none 한 번에 1개의 프로세스 실행]



하지만 한 번에 1개의 프로세스만 실행했을 때는 write와 read 모두 bfq 스케줄러 보다 none이 약 10% 만큼 더 빨랐습니다.

**2.1.1 bfq scheduler의 매개변수**

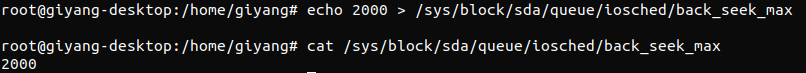


ls /sys/block/sda/queue/iosched

명령어로 bfq에서 사용 조정 가능한 매개변수 확인

**back\_seek\_max**

back\_seek\_max의 기본값은 16384입니다.



echo 를 이용해 역방향 검색을 위한 최대 거리를 KB 단위로 조정할 수 있습니다.

이 값은 현재 디스크의 헤드가 위치한 곳을 기준으로 backward seeking의 최대값을 의미합니다.

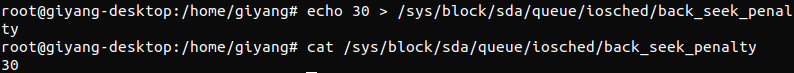
예를 들어 현재 헤드의 위치를 10이라고 하고 back\_seek\_max 값을 5라고 가정한다면 헤드의 위치 10에서 5를 뺀 위치 5번에서 9번의 요청들의 바로 다음 요청으로 간주되어 처리됩니다.

이 값은 인접한 위치의 요청이 들어왔을 때 바로 처리할 수 있게 되어 헤더의 움직임을 최소화하는데 도움이 되지만 그만큼 다른 요청들이 밀릴 수 있습니다.

**back\_seek\_penalty**

****

back\_seek\_penalty의 기본값은 2입니다.



이 매개변수는 역방향 탐색 비용을 계산하는데 사용합니다. 디스크의 헤드는 가능하다면 순서대로 움직이는 것이 좋은데, 예를 들어 헤드가 10번 위치에서 11번 위치와 9번 위치로 가는 요청이 있을때 둘 다 현재 헤드의 위치에서 1만큼 이동하지만 back\_seek\_penalty에 의해 back\_seek\_penalty의 값이 곱해져서, 9번 위치로 가는 요청이 11번 위치로 가는 요청 보다 성능에 좋지 않은 영향을 주는 요청이라고 간주합니다.

**low\_latency**

I/O 요청을 처리하다가 발생할 수 있는 대기 시간을 줄이는 역할을 합니다. BFQ의 낮은 대기 시간 모드를 활성화(1) 또는 비활성화(0)할 수 있습니다. 기본적으로 낮은 대기 시간 모드가 활성화(1)되어 있습니다. 높은 처리량을 원한다면 이 매개변수를 비활성화 시키는 것이 좋습니다.



cat /sys/block/sda/queue/iosched/low\_latency

명령어로 low\_latency의 값을 확인해본 결과, 기본값은 1이었습니다.



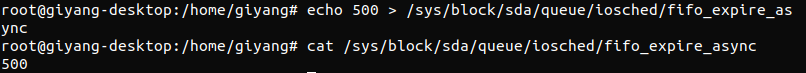
echo 0 > /sys/block/sda/queue/iosched/low\_latency

명령어로 low\_latency의 값을 1 또는 0으로 설정할 수 있습니다.

**fifo\_expire\_async**

****

fifo\_expire\_async의 기본값은 250입니다.

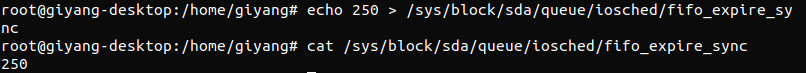


이 매개변수는 비동기 요청의 종료 시간을 설정하는데 사용합니다. 단위는 ms로 설정되어 있습니다. async는 비동기적이라는 뜻으로 주로 쓰기 요청을 나타냅니다.

**fifo\_expire\_sync**

****

fifo\_expire\_sync의 기본값은 125입니다.



fifo\_expire\_sync는 동기 요청의 종료 시간을 설정하는데 사용하고, 주로 읽기 요청을 나타냅니다. fifo\_expire\_async와 마찬가지로 단위는 ms로 설정되어 있습니다. 비동기식 요청보다 동기식 요청을 선호하면 이 매개변수의 값을 fifo\_expire\_async의 값 보다 낮게 설정해야됩니다.

**max\_budget**

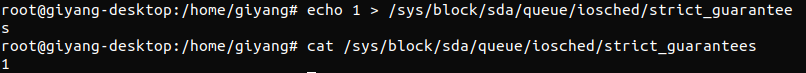
max\_budget의 기본값은 0입니다.

****

max\_budget의 값이 클수록 순차 I/O 요청 비율에 비례하여 처리량이 증가합니다.

**strict\_guarantees**

strict\_guarantees의 기본값은 0입니다.

****

이 매개변수를 활성화 하면, I/O 요청 대기열이 비어 있을 때 유휴 상태가 되고, 강제로 한 번에 하나의 I / O 요청을 처리하도록 합니다.

**timeout\_sync**

timeout\_sync의 기본값은 124입니다.



선택된 I/O 작업에 부여할 수 있는 최대 시간입니다. 비용이 많이 드는 장치를 찾을 때, 이 시간을 늘리면 최대 처리량이 증가합니다.

**slice\_idle**

slice\_idle의 기본값은 8입니다.



이 매개변수는 특정 동기화 bfq 대기열이 비어있을 때 다음 I/O 요청을 위해 bfq를 유휴 상태로 유지하는 시간을 지정합니다. 기본 단위는 ms입니다. 현재 요청중인 I/O를 모두 처리하고나서, 다음 I/O 요청이 들어올 수 있기 때문에 대기하고, I/O 요청이 들어온다면 해당 요청을 처리합니다. 이렇게 하면 디스크 헤드의 움직임을 최소화할 수 있습니다.

**slice\_idle\_us**

slice\_idle의 값을 마이크로 초로 표현한 것입니다. slice\_idle의 값을 변경하면 이 매개변수의 값도 자동으로 변경됩니다.