Red Hat Enterprise Linux 6

자원 관리 가이드

Red Hat Enterprise Linux 6에서 시스템 자원을 관리

엮음 1

Last Updated: 2017-10-12

Red Hat Enterprise Linux 6 자원 관리 가이드

Red Hat Enterprise Linux 6에서 시스템 자원을 관리
엮음 1

Martin Prpič
Red Hat Engineering Content Services
mprpic@redhat.com

Rüdiger Landmann
Red Hat Engineering Content Services
r.landmann@redhat.com

Douglas Silas
Red Hat Engineering Content Services
dhensley@redhat.com

법적 공지

초록

Red Hat Enterprise Linux 6에서 시스템 자원을 관리

차례

1장. 컨트롤 그룹 (CGROUPS) 소개

Red Hat Enterprise Linux 6에서는 *컨트롤 그룹 (control group)*이라는 새로운 커널 기능을 제공합니다. 이 문서에서는 이 기능을 *cgroup*라는 약칭으로 기재하고 있습니다. Cgroup을 통해 사용자는 CPU 시간, 시스템 메모리, 네트워크 대역폭과 같은 자원이나 이러한 자원의 조합을 시스템에서 실행 중인 사용자 정의 작업 그룹 (프로세스) 간에 할당할 수 있습니다. 또한 설정한 cgroup을 모니터링하거나 특정 자원으로의 cgroup 액세스를 거부하는 것 이외에 실행 중인 시스템에서 cgroup을 동적으로 다시 구성할 수 있습니다. **cgconfig** ("*control group config*") 서비스는 부팅시 시작 및 사전 정의된 cgroup을 다시 구성하도록 설정하여 재부팅 후에도 구성된 사항이 지속되도록 설정할 수 있습니다.

cgroup을 사용하여 시스템 관리자는 시스템 자원 할당, 우선 순위 지정, 거부, 관리, 모니터링과 같은 세밀한 제어가 가능합니다. 하드웨어 자원은 작업 및 사용자 간을 신속하게 분배하여 전체적인 효율성을 향상시킬 수 있습니다.

1.1. 컨트롤 그룹의 구성 방법

프로세스와 마찬가지로 cgroup은 계층적으로 구성되어 있으며 하위 cgroup 부모 cgroup 속성의 일부를 상속하도록되어 있습니다. 그러나이 두 모델 사이에는 차이점이 있습니다.

Linux 프로세스 모델

Linux 시스템의 모든 프로세스는 일반적인 부모 프로세스의 자식 프로세스입니다. **init** 프로세스는 부팅시 커널에 의해 실행되어 다른 프로세스를 시작합니다 (그 결과 자식의 자식 프로세스를 시작하는 경우도 있습니다). 모든 프로세스는 하나의 부모 프로세스의 하위 프로세스이기 때문에 Linux 프로세스 모델은 단일 계층 또는 트리로 되어 있습니다.

또한 **init**을 제외한 모든 Linux 프로세스는 환경 (예: PATH 변수) [1] 및 부모 프로세스의 기타 다른 속성 (예: 파일 열기 설명자)을 상속합니다.

Cgroup 모델

Cgroup은 다음과 같은 점에서 프로세스와 유사합니다:

- 계층적이다.
- 자식 cgroup은 부모 cgroup에서 특정 속성을 상속한다.

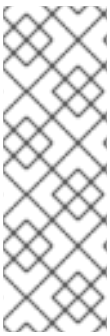
근본적인 차이점은 여러 다른 cgroup 계층이 시스템에 동시에 존재할 수 있다는 것입니다. Linux 프로세스 모델이 단일 프로세스 트리일 경우, cgroup 모델은 하나의 또는 여러 다른 연결되지 않은 작업 트리 (즉 프로세스)라는 것입니다.

각 계층에는 *하나 이상의 서브시스템*에 연결되므로 cgroup에 대해 여러 분리된 계층이 필요합니다. 서브시스템 [2]은 CPU 시간 또는 메모리와 같은 단일 자원을 말합니다. Red Hat Enterprise Linux 6는 9 개의 cgroup 서브시스템을 제공하고 있으며 그 이름과 기능은 다음과 같습니다.

Red Hat Enterprise Linux에서 사용 가능한 서브시스템

- **blkio** — 이 서브시스템은 물리 드라이브 (예: 디스크, 솔리드 스테이트, USB 등)와 같은 블록 장치에 대한 입력/출력 액세스에 제한을 설정합니다.
- **cpu** — 이 서브시스템은 CPU에 cgroup 작업 액세스를 제공하기 위해 스케줄러를 사용합니다.
- **cpuacct** — 이 하위 시스템은 cgroup의 작업에 사용된 CPU 자원에 대한 보고서를 자동으로 생성합니다.
- **cpuset** — 이 서브시스템은 개별 CPU (멀티코어 시스템에서) 및 메모리 노드를 cgroup의 작업에 할당합니다.

- **devices** — 이 서브시스템은 cgroup의 작업 단위로 장치에 대한 액세스를 허용하거나 거부합니다.
- **freezer** — 이 서브시스템은 cgroup의 작업을 일시 중지하거나 다시 시작합니다.
- **memory** — 이 서브시스템은 cgroup의 작업에서 사용되는 메모리에 대한 제한을 설정하고 이러한 작업에서 사용되는 메모리 자원에 대한 보고서를 자동으로 생성합니다.
- **net_cls** — 이 서브시스템은 Linux 트래픽 컨트롤러 (**tc**)가 특정 cgroup 작업에서 발생하는 패킷을 식별하게 하는 클래식 식별자 (classid)를 사용하여 네트워크 패킷에 태그를 지정합니다.
- **ns** — *namespace* 서브시스템



참고

man 페이지와 커널 문서 와 같은 cgroup 관련 문서에서 *자원 컨트롤러* 또는 *컨트롤러*라는 용어가 사용되는 경우가 있습니다. 이러한 두 용어는 "서브시스템 (subsystem)"과 같은 의미로 서브시스템이 정상적으로 리소스를 스케줄하거나 연결된 계층에 있는 cgroup에 제한을 적용하는 이유에서 이렇게 불리고 있습니다.

서브시스템 (자원 컨트롤러)의 정의는 매우 일반적인 것으로 작업 그룹 (즉 프로세스)에 따라 동작하는 것입니다.

1.2. 서브시스템, 계층, 컨트롤 그룹, 작업 간의 관계

cgroup 용어에서는 시스템 프로세스는 작업이라는 것을 염두에 두십시오.

그럼 여기서 서브시스템, cgroup 계층 및 작업 간의 관계를 관리하는데 있어서의 몇 가지 간단한 규칙과 그 규칙에 따른 결과에 대해 살펴보겠습니다.

규칙 1

단일 서브시스템 (예: **cpu**)는 하나의 계층에만 연결할 수 있습니다.

*결과적으로 **cpu** 서브시스템은 두 개의 서로 다른 계층에 연결될 수 없습니다.*

규칙 2

단일 계층에 하나 이상의 서브시스템을 연결할 수 있습니다.

*결과적으로 **cpu** 및 **memory** 서브시스템 (혹은 서브시스템 수)을 각각이 다른 계층에 연결되어 있지 않는 한 단일 계층에 연결할 수 있습니다.*

규칙 3

시스템에 새로운 계층이 생성될 때 마다 시스템의 모든 작업은 해당 계층의 디폴트 cgroup의 멤버가 됩니다. 이는 *root cgroup*이라고 합니다. 생성된 단일 계층의 경우 시스템의 각 작업은 해당 계층에 있는 *정확하게 하나의* cgroup의 멤버가 될 수 있습니다. 각각의 cgroup이 다른 계층에 있는 한 단일 작업은 여러 cgroup에 있을 수 있습니다. 작업이 동일한 계층에 있는 두 번째 cgroup의 멤버가 될 경우, 이는 계층의 첫 번째 cgroup에서 바로 삭제됩니다. 작업이 동일한 계층에 있는 두 개의 다른 cgroup의 멤버가 될 수 없습니다.

*결과적으로 **cpu** 및 **memory** 서브시스템이 **cpu_and_mem**이라는 계층에 연결되고 **net_cls** 서브시스템이 **net**이라는 계층에 연결된 경우, 실행 중인 **httpd** 프로세스는 **cpu_and_mem**에 있는 하나의 cgroup 그리고 **net**에 있는 하나의 cgroup의 멤버가 될 수 있습니다.*

`httpd` 프로세스가 속한 `cpu_and_mem`에 있는 cgroup은 다른 프로세스에 할당된 시간의 절반으로 CPU 시간이 제한되어 있고 메모리 사용량이 최대 `1024` MB로 제한되는 멤버일 수 있습니다. 또한, `net`에 있는 cgroup은 전송 속도가 초당 `30` 메가바이트로 제한되는 멤버일 수 있습니다.

첫 번째 계층이 생성되면 시스템의 모든 작업은 최소 한 개의 cgroup (root cgroup)의 멤버가 됩니다. 따라서 cgroup을 사용하면 모든 시스템 작업은 최소 하나의 cgroup에 있게 됩니다.

규칙 4

자신을 포크하는 시스템에 있는 하나의 프로세스 (작업)는 자식 프로세스(작업)을 생성합니다. 자식 작업은 자동으로 부모가 멤버인 모든 cgroup의 멤버가 됩니다. 그 후 자식 작업은 필요에 따라 다른 cgroup으로 이동할 수 있지만 처음에는 부모 작업의 cgroup (프로세스 용어로 "환경"이라함)을 상속합니다.

결과적으로 `cpu_and_mem` 계층에 있는 `half_cpu_1gb_max`라는 cgroup의 멤버와 `net` 계층에 있는 `trans_rate_30`라는 cgroup의 멤버인 `httpd` 작업을 고려해 봅시다. `httpd` 프로세스가 자신을 포크하면 자식 프로세스는 자동으로 `half_cpu_1gb_max` cgroup과 `trans_rate_30` cgroup의 멤버가 되어 부모 작업이 속해 있는 것과 동일한 cgroup을 상속합니다.

이후에는 부모 작업 및 자식 작업이 완전히 서로 독립적인 상태가 되어 하나의 작업이 속한 cgroup을 변경해도 다른 작업에 영향을 미치지 않습니다. 또한 부모 작업의 cgroup을 변경해도 손자 작업에도 전혀 영향을 주지 않습니다. 즉, 자식 작업은 모두 처음에는 부모 작업과 동일한 cgroup에 있는 멤버쉽을 상속하지만 이러한 멤버쉽은 나중에 변경 또는 삭제할 수 있습니다.

1.3. 자원 관리의 의미

- 작업은 하나의 계층 구조에서 하나의 단일 cgroup에만 속할 수 있기 때문에 한 가지 방식만이 단일 서브시스템에 의해 제한받거나 영향을 받을 수 있습니다. 이는 제한이 아니라 기능이며 논리에 맞습니다.
- 단일 계층의 모든 작업에 영향을 미치는 여러 서브시스템을 함께 그룹화할 수 있습니다. 계층 구조에 있는 cgroup은 다른 매개 변수가 설정되어 있기때문에 이러한 작업에 미치는 영향이 달라집니다.
- 경우에 따라 계층을 *리팩토링*할 필요가 있을 수 있습니다. 예를 들어, 여러 서브시스템이 연결된 계층에서 서브시스템을 제거하거나 이를 새로운 다른 계층에 연결하는 경우입니다.
- 반대로 말하면 별도의 계층 사이에서 서브시스템을 분할해야 할 필요성이 감소하면 계층을 제거하기 위해 서브시스템을 기존 계층에 연결할 수 있다는 것입니다.
- 이는 cpu와 메모리 서브시스템이 연결되어 있는 것과 같은 단일 계층의 특정 작업에 대한 몇몇 매개 변수를 설정하는 등 단순한 cgroup사용을 가능하게 합니다.
- 이는 고도로 특화된 설정도 가능하게 하여, 시스템의 각 작업 (프로세스)은 하나의 서브시스템이 연결된 각 계층의 멤버가 될 수 있습니다. 이러한 설정으로 시스템 관리자는 모든 작업의 모든 매개 변수를 전적으로 관리할 수 있게 됩니다.

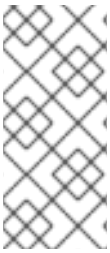
---

[1] 부모 프로세스는 자식 프로세스에게 전달하기 전 환경을 변경하는 것이 가능.

[2] 서브시스템은 libcgroup man 페이지 및 기타 다른 문서에서 *자원 컨트롤러*, 또는 단순히 *컨트롤러*라고도 부릅니다.

2장. 컨트롤 그룹 사용하기

cgroup를 사용하여 작업하는 가장 간단한 방법은 libcgroup 패키지를 설치하는 것입니다. 이 패키지에는 여러 cgroup 관련 명령행 유틸리티와 man 페이지가 포함되어 있습니다. 모든 시스템에서 사용 가능한 쉘 명령어와 유틸리티를 사용하여 계층을 *마운트*하고 cgroup 매개 변수를 (비 영구적으로) 설정하는 것도 가능하지만, libcgroup 제공 유틸리티를 사용하면 프로세스를 단순화하고 기능을 확장할 수 있습니다. 따라서 이 문서에서는 libcgroup 명령에 중점을 두어 설명하겠습니다. 대부분의 경우 기본 구조를 설명하기 위해 해당 쉘 명령을 포함시키지만 실제로 libcgroup 명령을 사용하는 것이 좋습니다.



참고

cgroup을 사용하려면 root로 다음 명령을 실행하여 libcgroup 패키지가 설치되어 있는지 확인합니다:

```
~]# yum install libcgroup
```

2.1. CGCONFIG 서비스

libcgroup 패키지와 함께 설치되는 **cgconfig** 서비스는 계층 구조를 생성하여 서브시스템을 연결하고 계층 내에서 cgroup을 관리하기 위한 편리한 방법을 제공합니다. 시스템에서 계층과 cgroup을 관리하기 위해 **cgconfig**를 사용하는 것이 좋습니다.

cgconfig 서비스는 Red Hat Enterprise Linux 6에서 기본값으로 시작되지 않습니다. **chkconfig**로 서비스를 시작할 때, 이는 cgroup 설정 파일 — **/etc/cgconfig.conf**을 읽습니다. 따라서 Cgroup은 시스템 시작 시 마다 다시 생성되어 영구적으로 됩니다. 설정 파일의 내용에 따라 **cgconfig**는 계층 구조를 생성하고 필요한 파일 시스템을 마운트하여 cgroup을 만들고 각 그룹에 대해 서브시스템 매개 변수를 설정합니다.

libcgroup 패키지로 설치된 디폴트 **/etc/cgconfig.conf** 파일은 각 서브시스템에 대해 개별적 계층 구조를 생성하여 마운트하고 이 계층구조에 서브시스템을 연결합니다.

cgconfig 서비스를 중지 (**service cgconfig stop** 명령을 사용)하면, 마운트된 모든 계층을 마운트 해제합니다.

2.1.1. cgconfig.conf 파일

/etc/cgconfig.conf 파일에는 두 가지 주요한 항목 *mount* 및 *group*이 있습니다. 마운트 항목은 가상 파일 시스템으로 계층구조를 생성하고 마운트하며 이러한 계층 구조에 서브시스템을 연결합니다. 마운트 항목은 다음과 같은 구문을 사용하여 정의됩니다:

```
mount {
    <controller> = <path>;
    …
}
```

사용 예제는 예 2.1. “마운트 항목 생성하기 ”에서 참조하십시오.

예 2.1. 마운트 항목 생성하기

다음의 예에서는 **cpuset** 서브시스템에 대한 계층 구조를 생성하고 있습니다:

```
mount {
    cpuset = /cgroup/cpu;
}
```

이에 해당하는 쉘 명령은 다음과 같습니다:

```
~]# mkdir /cgroup/cpu
~]# mount -t cgroup -o cpu cpu /cgroup/cpu
```

그룹 항목은 cgroup을 생성하고 서브시스템 매개 변수를 설정합니다. 그룹 항목은 다음과 같은 구문을 사용하여 정의됩니다:

```
group <name> {
    [<permissions>]
    <controller> {
        <param name> = <param value>;
        …
    }
    …
}
```

permissions 섹션은 옵션이라는 점에 유의하십시오. 그룹 항목의 권한을 정의하려면 다음과 같은 구문을 사용합니다:

```
perm {
    task {
        uid = <task user>;
        gid = <task group>;
    }
    admin {
       uid = <admin name>;
       gid = <admin group>;
    }
}
```

사용 예제는 예 2.2. “그룹 항목 생성하기 ”에서 참조하십시오.

예 2.2. 그룹 항목 생성하기

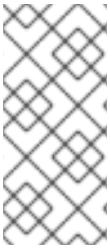
다음 예에서는 sql 데몬에 대한 cgroup을 생성하고 **sqladmin** 그룹의 사용자에게 cgroup에 작업을 추가할 수 있는 권한을 부여하며 **root** 사용자에게 서브시스템 매개 변수를 변경할 수 있는 권한을 부여합니다:

```
group daemons/sql {
    perm {
        task {
            uid = root;
            gid = sqladmin;
        } admin {
            uid = root;
            gid = root;
        }
    } cpu {
```

```
        cpu.shares = 100;
    }
}
```

예 2.1. "마운트 항목 생성하기 "에 있는 마운트 항목의 예제와 조합할 때 해당하는 쉘 명령은 다음과 같습니다:

```
~]# mkdir -p /cgroup/cpu/daemons/sql
~]# chown root:root /cgroup/cpu/daemons/sql/*
~]# chown root:sqladmin /cgroup/cpu/daemons/sql/tasks
~]# echo 100 > /cgroup/cpu/daemons/sql/cpu.shares
```



참고

/etc/cgconfig.conf에 있는 변경 사항을 활성화하려면 **cgconfig** 서비스를 다시 시작해야 합니다:

```
~]# service cgconfig restart
```

libcgroup 패키지를 설치할 때, 예제 설정 파일은 **/etc/cgconfig.conf**에 작성됩니다. 각 줄의 맨 앞에 해시 기호 ('#')가 있는 경우 그 행은 주석처리되어 **cgconfig** 서비스가 해석되지 않도록 합니다.

2.2. 계층 생성 및 서브시스템 연결

주의

다음의 새로운 계층 구조 생성 및 서브시스템으로의 연결 지침에서는 시스템에서 cgroup이 아직 설정되어 있지 않은 것을 전제로 하고 있습니다. 이러한 경우, 이 단계는 시스템의 작동에 영향을 미치지 않습니다. 하지만 작업이 있는 cgroup의 조정 가능한 매개 변수를 변경하면 해당 작업에 즉시 영향을 미칠 수 있습니다. 이 문서에서는 하나 이상의 작업에 영향을 미칠 수 있는 조정 가능한 cgroup 매개 변수를 처음으로 변경 할 때 경고합니다.

cgroup이 이미 설정된 (수동 또는 **cgconfig** 서비스를 사용하여) 시스템에서 기존 계층을 마운트 해제하지 않으면 이러한 명령은 실패하여 시스템의 작동에 영향을 미칩니다. 프로덕션 시스템에서 이 절차를 시도하지 마십시오.

계층 구조를 생성하고 서브시스템을 이에 연결하려면 root로 **/etc/cgconfig.conf** 파일의 **mount** 섹션을 편집합니다. **mount** 섹션에 있는 항목은 다음과 같은 형식입니다:

```
subsystem = /cgroup/hierarchy;
```

cgconfig가 다음 부팅 때 계층을 만들고 이에 서브시스템을 연결합니다.

다음 예제에서는 **cpu_and_mem**라는 계층을 만들고 이에 **cpu**, **cpuset**, **cpuacct**, **memory** 서브시스템을 연결합니다.

```
mount {
    cpuset  = /cgroup/cpu_and_mem;
    cpu     = /cgroup/cpu_and_mem;
    cpuacct = /cgroup/cpu_and_mem;
    memory  = /cgroup/cpu_and_mem;
}
```

다른 방법

쉘 명령 및 유틸리티를 사용하여 계층을 생성하고 서브시스템을 연결할 수 있습니다.

root로 계층의 *mount point*를 생성합니다. 마운트 포인트에는 cgroup의 이름이 포함되어 있어야 합니다:

```
~]# mkdir /cgroup/name
```

예를 들어:

```
~]# mkdir /cgroup/cpu_and_mem
```

다음으로 **mount** 명령을 사용하여 계층을 마운트하고 동시에 하나 이상의 서브시스템을 연결합니다. 예:

```
~]# mount -t cgroup -o subsystems name /cgroup/name
```

여기서 *subsystems*은 콤마로 구분된 서브시스템 목록이고 *name*은 계층 이름입니다. 사용 가능한 모든 서브시스템에 대한 간단한 설명은 Red Hat Enterprise Linux에서 사용 가능한 서브시스템 에 기재되어 있으며 3장. *서브시스템 및 조정 가능한 매개 변수* 에는 자세한 내용이 설명되어 있습니다.

예 2.3. mount 명령을 사용하여 서브시스템을 연결

이 예제에서는 **/cgroup/cpu_and_mem**라는 디렉토리가 이미 존재하여 생성된 계층의 마운트 지점으로 작동합니다. **cpu**, **cpuset** 및 **memory** 서브시스템을 계층에 연결하고 **cpu_and_mem**으로 이름 지정한 후 **/cgroup/cpu_and_mem**에 있는 **cpu_and_mem** 계층에 **mount**합니다:

```
~]# mount -t cgroup -o cpu,cpuset,memory cpu_and_mem /cgroup/cpu_and_mem
```

현재 마운트 지점 (즉, 연결된 계층이 마운트된 위치)과 함께 사용 가능한 모든 서브시스템을 나열하기 위해 **lssubsys** 명령 [3]을 사용할 수 있습니다.

```
~]# lssubsys -am
cpu,cpuset,memory /cgroup/cpu_and_mem
net_cls
ns
cpuacct
devices
freezer
blkio
```

이 출력 결과는 다음과 같은 내용을 나타냅니다:

- **cpu**, **cpuset** 및 **memory** 서브시스템은 **/cgroup/cpu_and_mem**에 마운트된 계층에 연결되어 있습니다.

- **net_cls**, **ns**, **cpuacct**, **devices**, **freezer** 및 **blkio** 서브시스템은 해당 마운트 지점이 없기 때문에 아직 어떤 계층에도 연결되어 있지 않습니다.

2.3. 기존 계층에 서브시스템 연결 및 연결 해제

서브시스템을 기존 계층에 추가하려면 기존 계층에서 연결 해제하거나 다른 계층으로 이동해서 root로 **/etc/cgconfig.conf** 파일의 **mount** 섹션을 2.2절. "계층 생성 및 서브시스템 연결"에 기재되어 있는 것과 동일한 구문을 사용하여 편집합니다. **cgconfig**는 다음 부팅 시 지정한 계층에 따라 서브시스템을 다시 구성하게 됩니다.

다른 방법

연결되지 않은 서브시스템을 기존 계층에 추가하려면 계층을 다시 마운트합니다. **remount** 옵션과 함께 **mount** 명령에 추가 서브시스템을 포함시킵니다.

예 2.4. 계층 구조를 다시 마운트하여 서브시스템을 추가

lssubsys 명령은 **cpu_and_mem** 계층 구조에 부착된 **cpu**, **cpuset**, **memory**를 보여줍니다.

```
~]# lssubsys -am
cpu,cpuset,memory /cgroup/cpu_and_mem
net_cls
ns
cpuacct
devices
freezer
blkio
```

remount 옵션을 사용하여 서브시스템 목록에 **cpuacct**를 포함하여 **cpu_and_mem** 계층을 다시 마운트합니다:

```
~]# mount -t cgroup -o remount,cpu,cpuset,cpuacct,memory cpu_and_mem
/cgroup/cpu_and_mem
```

lssubsys 명령을 실행하면 **cpu_and_mem** 계층에 부착된 **cpuacct**가 나타납니다:

```
~]# lssubsys -am
cpu,cpuacct,cpuset,memory /cgroup/cpu_and_mem
net_cls
ns
devices
freezer
blkio
```

유사하게 계층 구조를 다시 마운트하고 서브시스템 이름을 -o 옵션에서 제거하여 기존 계층에서 서브시스템 연결을 해제할 수 있습니다. 예를 들어, **cpuacct** 서브시스템의 연결을 해제하려면 단순히 다시 마운트하고 이름을 삭제합니다:

```
~]# mount -t cgroup -o remount,cpu,cpuset,memory cpu_and_mem
/cgroup/cpu_and_mem
```

2.4. 계층 마운트 해제

umount 명령을 실행하여 cgroup의 계층을 *unmount*할 수 있습니다:

```
~]# umount /cgroup/name
```

예를 들어:

```
~]# umount /cgroup/cpu_and_mem
```

계층이 현재 비어 있을 경우 (즉, root cgroup 만을 포함하는 경우)에는 계층은 마운트 해제시 비활성화됩니다. 계층에 다른 cgroup이 있을 경우 계층은 마운트되지 않더라도 커널에서 활성 상태를 유지합니다.

계층을 제거하려면 계층을 마운트 해제하기 전 모든 자식 cgroup이 삭제되었는지 확인합니다. 또는 **cgclear** 명령을 사용하면 비어 있지 않은 계층도 비활성화할 수 있습니다 — 2.11절. "컨트롤 그룹 업로드 " 참조.

2.5. 컨트롤 그룹 만들기

cgcreate 명령을 사용하여 cgroup을 생성합니다. **cgcreate**의 구문은 **cgcreate -t *uid*:*gid* -a *uid*:*gid* -g *subsystems*:*path*** 입니다. 여기서:

- **-t** (옵션) — 이 cgroup에 대해 **tasks** 가상 파일을 소유하는 사용자 (사용자 ID인 uid로 지정)와 그룹 (그룹 ID인 gid로 지정)을 지정합니다. 이 사용자는 cgroup에 작업을 추가할 수 있습니다.

  참고

  cgroup에서 작업을 제거하려면 다른 cgroup로 이를 이동하는 것이 유일한 방법임에 유의하십시오. 작업을 이동하려면 사용자는 *대상* cgroup에 쓰기 권한이 있어야 합니다. 소스 cgroup에 대한 쓰기 액세스는 중요하지 않습니다.

- **-a** (옵션) — 이 cgroup에 대해 **tasks** 이외의 모든 가상 파일을 소유하는 사용자 (사용자 ID인 uid로 지정)와 그룹 (그룹 ID인 gid로 지정)을 지정합니다. 이 사용자는 cgroup에 있는 작업의 시스템 자원에 대한 액세스를 변경할 수 있습니다.

- **-g** — cgroup이 생성되어야 하는 계층을 계층에 관련된 콤마로 구분된 *subsystems* 목록으로 지정합니다. 이 목록의 서브시스템이 다른 계층에 있는 경우, 그룹은 각 계층에 생성됩니다. 계층 목록은 콜론 및 계층에 관련된 자식 그룹으로 *path*로 계속됩니다. 경로에 계층의 마운트 지점을 포함시키지 마십시오.

  예를 들어, **/cgroup/cpu_and_mem/lab1/** 디렉토리에 위치한 cgroup은 **lab1**라고 부릅니다. 서브시스템에 계층이 하나 밖에 없기 때문에 경로는 이미 고유하게 식별되는 것입니다. 또한 그룹은 cgroup이 생성된 계층에 있는 모든 서브시스템에 의해 제어된다는 점에 유의하십시오. **cgcreate** 명령에서 이러한 서브시스템이 지정되지 않은 경우에도 그러합니다 — 예 2.5. "cgcreate 사용법 " 참조.

동일한 계층에 있는 모든 cgroup은 모두 동일한 컨트롤러를 가지고 있기 때문에 자식 그룹은 부모와 동일한 컨트롤러를 갖게 됩니다.

예 2.5. cgcreate 사용법

cpu_and_mem 계층에서 **cpu** 및 **memory** 서브시스템이 함께 마운트되어 있으며 **net_cls** 컨트롤러가 **net**라는 다른 계층에 마운트되어 있는 시스템을 가정하여 다음 명령을 실행합니다:

```
~]# cgcreate -g cpu,net_cls:/test-subgroup
```

cgcreate 명령은 **test-subgroup**라는 두 개의 그룹을 생성합니다. 하나는 **cpu_and_mem** 계층에 다른 하나는 **net** 계층에 들어갑니다. **cgcreate** 명령으로 지정하지 않아도 **cpu_and_mem** 계층에 있는 **test-subgroup** 그룹은 **memory** 서브시스템에 의해 제어됩니다.

다른 방법
cgroup의 자식을 생성하려면 **mkdir** 명령을 사용합니다:

```
~]# mkdir /cgroup/hierarchy/name/child_name
```

예를 들어:

```
~]# mkdir /cgroup/cpuset/lab1/group1
```

2.6. 컨트롤 그룹 삭제하기

cgcreate와 유사한 구문을 갖는 **cgdelete**로 cgroup을 제거합니다. **cgdelete** ***subsystems*:*path***를 실행합니다. 여기서:

- *subsystems*은 콤마로 구분되는 서브시스템 목록입니다.
- *path*는 계층 구조의 root와 관련된 group으로의 경로입니다.

예를 들어:

```
~]# cgdelete cpu,net_cls:/test-subgroup
```

cgdelete에서 **-r** 옵션을 사용하여 모든 하위 그룹을 재귀적으로 삭제할 수 있습니다.

cgroup을 삭제하면, 해당 cgroup 작업은 부모 그룹으로 이동합니다.

2.7. 매개변수 설정하기

해당 cgroup을 수정하기 위해 권한을 갖는 사용자 계정에서 **cgset** 명령을 실행하여 서브시스템 매개 변수를 설정합니다. 예를 들어, **/cgroup/cpuset/group1**이 존재하는 경우 다음과 같은 명령으로 그룹에 액세스할 수 있는 CPU를 지정합니다:

```
cpuset]# cgset -r cpuset.cpus=0-1 group1
```

cgset의 구문은 **cgset -r *parameter=value path_to_cgroup*** 입니다. 여기서:

- *parameter*는 설정해야 할 매개 변수로 특정 group의 디렉토리에 있는 파일에 해당합니다.
- *값*은 매개변수에 지정할 값입니다
- *path_to_cgroup*는 *계층의 root에 관련된* cgroup으로의 경로입니다. 예를 들어, root 그룹의 매개 변수를 설정하려면 (**/cgroup/cpuacct/**가 존재하는 경우) 다음 명령을 실행합니다:

  ```
  cpuacct]# cgset -r cpuacct.usage=0 /
  ```

또한 **.**은 root 그룹과 관련되어 있기 때문에 (즉, root 그룹 자체이므로) 다음 명령을 실행할 수 있습니다:

```
cpuacct]# cgset -r cpuacct.usage=0 .
```

하지만 **/**는 권장 구문이라는 점에 유의하십시오.

참고

root 그룹에 소수의 매개 변수 만을 설정할 수 있습니다 (예: 위의 예제에서 **cpuacct.usage** 매개 변수 등). root 그룹이 기존의 모든 자원을 소유하기 때문입니다. 따라서 **cpuset.cpu** 매개 변수와 같은 특정 매개 변수를 정의하여 모든 기존 프로세스를 제한하는 것은 의미가 없습니다.

root 그룹의 하위 그룹인 **group1**의 매개 변수를 설정하려면 다음 명령을 실행합니다:

```
cpuacct]# cgset -r cpuacct.usage=0 group1
```

그룹 이름의 끝에 있는 슬래시 (예: **cpuacct.usage=0 group1/**)는 옵션입니다.

cgset으로 설정할 수 있는 값은 특정 계층의 상위 계층에 설정된 값에 따라 달라집니다. 예를 들어, **group1**이 시스템에서 CPU 0만 사용하도록 제한된 경우, CPUs 0과 1 또는 CPU 1 만을 사용하도록 **group1/subgroup1**을 설정할 수 없습니다.

cgset을 사용하여 서로 다른 cgroup 간의 매개 변수를 복사할 수 있습니다. 예:

```
~]# cgset --copy-from group1/ group2/
```

cgset으로 매개 변수를 복사하기 위한 구문은 다음과 같습니다: **cgset --copy-from *path_to_source_cgroup path_to_target_cgroup*** 여기서:

- *path_to_source_cgroup*은 매개 변수가 복사된 계층의 root 그룹과 관련된 cgroup으로의 경로입니다.
- *path_to_target_cgroup*은 계층의 root 그룹과 관련하여 대상 cgroup으로의 경로입니다.

한 그룹에서 다른 그룹으로 매개 변수를 복사하기 전 다양한 서브시스템의 필수 매개 변수가 설정되어 있는지 확인합니다. 필수 매개 변수가 설정되지 않은 경우 명령은 실패하게 됩니다. 필수 매개 변수에 대한 자세한 내용은 중요 — 필수 매개 변수 에서 참조하십시오.

다른 방법

cgroup에서 매개 변수를 설정하려면 **echo** 명령을 사용하여 관련 서브시스템 가상 파일로 값을 추가합니다. 예를 들어, 다음 명령은 **0-1** 값을 cgroup **group1**의 **cpuset.cpus** 가상 파일에 추가합니다:

```
~]# echo 0-1 > /cgroup/cpuset/group1/cpuset.cpus
```

이 값이 지정되면 cgroup 내의 작업은 시스템의 CPU 0와 1에서만 수행되도록 제한됩니다.

2.8. 프로세스를 컨트롤 그룹으로 옮기기

cgclassify 명령을 실행하여 프로세스를 cgroup으로 옮길 수 있습니다:

```
~]# cgclassify -g cpu,memory:group1 1701
```

cgclassify의 구문은 **cgclassify -g *subsystems*:*path_to_cgroup* *pidlist***입니다. 여기서:

- *subsystems*은 콤마로 구분된 서브시스템 목록이거나 또는 사용 가능한 모든 서브시스템과 연결된 계층에 있는 프로세스를 시작하는 *****입니다. 동일한 이름의 cgroup이 여러 계층에 존재할 경우, **-g** 옵션을 지정하면 각 그룹에 프로세스가 생성되는 점에 유의하십시오. 여기서 지정된 서브시스템의 각 계층에 cgroup이 있는지 확인합니다.
- *path_to_cgroup*은 계층 구조내에서 cgroup으로의 경로입니다.
- *pidlist*는 *프로세스 식별자 (process identifier)* (PID)의 공백으로 구분된 목록입니다.

--sticky 옵션을 *pid* 앞에 추가하여 모든 자식 프로세스를 동일한 cgroup에 속하게 할 수 있습니다. 이 옵션을 설정하지 않고 **cgred** 데몬이 실행 중인 경우, 자식 프로세스는 **/etc/cgrules.conf**에 있는 설정에 따라 cgroup을 할당하게 됩니다. 하지만, 프로세스 자체는 시작한 cgroup에 남아있게 됩니다.

cgclassify를 사용하면 동시에 여러 프로세스를 옮길 수 있습니다. 예를 들어, 이 명령은 PID가 **1701** 및 **1138**인 프로세스를 cgroup **group1/**로 옮깁니다:

```
~]# cgclassify -g cpu,memory:group1 1701 1138
```

옮겨야 할 PID가 공백으로 구분되어 있으며 지정된 그룹은 다른 계층에 있어야 함에 유의하십시오.

다른 방법
프로세스를 cgroup으로 옮기기 위해서는 PID를 cgroup의 **tasks** 파일에 기록합니다. 예를 들어, PID **1701**로 프로세스를 **/cgroup/lab1/group1/**에 있는 cgroup으로 옮기려면 다음을 실행합니다:

```
~]# echo 1701 > /cgroup/lab1/group1/tasks
```

2.8.1. cgred 데몬

Cgred는 **/etc/cgrules.conf** 파일에 설정된 매개변수에 따라 작업을 cgroup으로 옮기는 데몬입니다. **/etc/cgrules.conf** 파일의 항목은 다음 두 형태중 하나입니다:

- *user hierarchies control_group*
- *user*:*command hierarchies control_group*

예를 들어:

```
maria   devices  /usergroup/staff
```

이 항목은 **maria**라는 사용자의 프로세스를 **/usergroup/staff** cgroup에 지정된 매개 변수에 따라 devices 서브시스템을 액세스하도록 지정합니다. 특정 명령을 특정 cgroup에 연결시키려면, 다음과 같이 *command* 매개 변수를 추가합니다:

```
maria:ftp  devices  /usergroup/staff/ftp
```

이 항목은 **maria**라는 사용자가 **ftp** 명령을 사용할 때는 **devices** 서브시스템이 들어있는 계층의 **/usergroup/staff/ftp** cgroup에 프로세스가 자동으로 이동하도록 지정합니다. 하지만 이 데몬은 해당 조건이 충족될 경우에만 프로세스를 cgroup로 이동한다는 점에 유의하십시오. 따라서 **ftp** 프로세스가

잘못된 그룹에서 짧은 시간 동안 실행될 수 있습니다. 또한 프로세스가 잘못된 그룹에 있는 동안 자식 프로세스가 발생한 경우, 이는 이동되지 않을 수 있습니다.

/etc/cgrules.conf 파일에 있는 항목에 다음과 같은 표기법을 추가할 수 있습니다:

- **@**문자가 *사용자*의 앞에 있으면, 개별 사용자가 아니라 그룹을 의미합니다. 예를 들어 **@admins**은 **admins** 그룹내에 있는 모든 사용자를 의미합니다.
- ***는 "전부"을 의미합니다. 예를 들어 **subsystem** 필드 안의 ***는 모든 서브시스템을 의미합니다.
- **%**는 한줄 위의 항목과 동일한 항목을 의미합니다. 예를 들어서:

```
@adminstaff  devices  /admingroup
@labstaff  %  %
```

2.9. 컨트롤 그룹 안에서 프로세스 시작하기

중요

일부 서브시스템은 이러한 서브시스템을 사용하는 cgroup으로 작업을 이전하기 전 반드시 설정해야 하는 매개 변수가 있습니다. 예를 들어, **cpuset** 서브시스템을 이용하는 cgroup으로 작업을 이전하기 전 cgroup에 대해 **cpuset.cpus** 및 **cpuset.mems** 매개 변수가 반드시 정의되어 있어야 합니다.

이 부분에 있는 예제에서는 명령에 대한 바른 문법을 보여주지만, 예제에서 사용되는 컨트롤러에 대한 필수 매개변수가 적절히 설정된 시스템에서만 제대로 작동할 것입니다. 만약 그러한 컨트롤러를 이미 설정하지 않은 경우라면, 예제에 있는 명령을 복사해서 실행해도 시스템에서 정상 동작하지 않을 것입니다.

특정 서브시스템에 필요한 매개 변수에 대한 설명은 3.10절. “추가 자원 ”에서 참조하십시오.

cgexec 명령을 실행해 cgroup에서 프로세스를 시작할 수 있습니다. 예를 들어, 이 명령은 **lynx** 웹 브라우저를 **group1** cgroup 내에서 실행하여 **cpu** 서브시스템에 의해 그룹에 부과되는 제한 사항이 적용되게 합니다:

```
~]# cgexec -g cpu:group1 lynx http://www.redhat.com
```

cgexec의 구문은 **cgexec -g *subsystems*:*path_to_cgroup* *command* *arguments*** 입니다. 여기서:

- *subsystems*은 콤마로 구분된 서브시스템 목록이거나 또는 사용 가능한 모든 서브시스템과 연결된 계층에 있는 프로세스를 시작하는 ***입니다. 2.7절. “매개변수 설정하기”에 설명된 **cgset** 처럼 동일한 이름의 cgroup이 여러 계층에 존재할 경우, **-g** 옵션을 지정하면 각 그룹에 프로세스가 생성되는 점에 유의하십시오. 여기서 지정된 서브시스템의 각 계층에 cgroup이 있는지 확인합니다.
- *path_to_cgroup*은 계층과 관련된 cgroup으로의 경로입니다.
- *command*는 실행할 명령입니다.
- *arguments*는 명령의 인수입니다

--sticky 옵션을 *command* 앞에 추가하여 자식 프로세스를 동일한 cgroup에 속하게 할 수 있습니다. 이 옵션을 설정하지 않고 **cgred** 데몬이 실행 중일 경우, 자식 프로세스는 **/etc/cgrules.conf**에서의 설정에 따라 cgroup을 할당하게 됩니다. 하지만 프로세스 자체는 시작한 cgroup에 남아있게 됩니다.

다른 방법
새로운 프로세스를 시작하면 해당 프로세스는 부모 프로세스의 그룹을 상속합니다. 따라서, 특정 cgroup 프로세스 시작을 위한 다른 방법은 쉘 프로세스를 그룹으로 이동하고 (2.8절. “프로세스를 컨트롤 그룹으로 옮기기” 참조) 그 쉘에서 프로세스를 시작하는 것입니다. 예:

```
~]# echo $$ > /cgroup/lab1/group1/tasks
lynx
```

lynx를 종료 후 기존 쉘은 **group1** cgroup에 있음에 유의하십시오. 따라서 더 나은 방법은 다음과 같습니다:

```
~]# sh -c "echo \$$ > /cgroup/lab1/group1/tasks && lynx"
```

2.9.1. 서비스를 컨트롤 그룹에서 시작하기

cgroup에 있는 특정 서비스를 시작할 수 있습니다. cgroup에서 시작할 수 있는 서비스는 다음 조건을 충족해야 합니다:

- **/etc/sysconfig/*servicename*** 파일을 사용합니다
- 서비스를 시작하기 위해 **/etc/init.d/functions**에서 **daemon()** 함수를 사용합니다.

서비스가 cgroup에서 시작되게 하려면, **/etc/sysconfig** 디렉토리에 있는 파일을 편집하여 **CGROUP_DAEMON="*subsystem*:*control_group*"** 형태의 항목을 추가시킵니다. 여기서 *subsystem*은 특정 계층과 관련된 서브시스템이고 *control_group*은 계층에 있는 cgroup입니다. 예:

```
CGROUP_DAEMON="cpuset:daemons/sql"
```

2.10. 컨트롤 그룹에 대한 정보 얻기

2.10.1. 프로세스 찾기

프로세스가 속한 컨트롤 그룹을 찾으려면, 다음을 실행합니다:

```
~]$ ps -O cgroup
```

만약 프로세스의 PID를 알고 있다면, 다음과 같이 합니다:

```
~]$ cat /proc/PID/cgroup
```

2.10.2. 서브시스템 찾기

커널에서 사용 가능한 서브시스템과 이러한 서브시스템이 어떻게 계층 구조로 마운트되는지를 확인하려면 다음 명령을 실행합니다:

```
~]$ cat /proc/cgroups
```

또는 특정 서브시스템의 마운트 지점을 확인하려면, 다음 명령을 실행합니다:

```
~]$ lssubsys -m subsystems
```

여기서 *subsystems*은 대상 서브시스템의 목록입니다. **lssubsys -m** 명령은 각 계층마다 최상위 마운트 지점만을 반환한다는 점에 유의하십시오.

2.10.3. 계층 확인

/cgroup 아래에 계층을 마운트할 것을 권장합니다. 사용하는 시스템이 이러한 경우에 해당한다고 가정할 경우, 계층 목록을 얻기 위해 디렉토리의 내용을 나열하거나 검색합니다. **tree**가 시스템에 설치되어 있을 경우 이를 실행하여 모든 계층과 계층내에 있는 cgroup 개요를 가져옵니다:

```
~]$ tree /cgroup/
```

2.10.4. 컨트롤 그룹 확인

시스템에 cgroup을 나열하려면 다음을 실행합니다:

```
~]$ lscgroup
```

controller*:*path 형식으로 컨트롤러와 경로를 지정하면 특정 계층에 출력을 제한할 수 있습니다. 예:

```
~]$ lscgroup cpuset:adminusers
```

cpuset 서브시스템이 연결된 계층에 있는 **adminusers** cgroup의 하위 그룹만을 나열합니다.

2.10.5. 컨트롤 그룹의 매개 변수 표시

특정 cgroup 매개 변수를 표시하려면 다음 명령을 실행합니다:

```
~]$ cgget -r parameter list_of_cgroups
```

여기서 *parameter*는 서브시스템의 값을 포함하는 가상 파일이며 *list_of_cgroups*는 공백으로 구분된 cgroup 목록입니다. 예:

```
~]$ cgget -r cpuset.cpus -r memory.limit_in_bytes lab1 lab2
```

cgroups **lab1** 및 **lab2**의 **cpuset.cpus** 및 **memory.limit_in_bytes** 값을 표시합니다.

매개 변수 자체의 이름을 모르는 경우에는 다음과 같은 명령을 사용합니다:

```
~]$ cgget -g cpuset /
```

2.11. 컨트롤 그룹 업로드

주의

cgclear 명령은 모든 계층에 있는 모든 cgroup을 제거합니다. 설정 파일에 이러한 계층 구조를 저장해 두지 않은 경우, 이를 바로 재구성할 수 없습니다.

전체 cgroup 파일 시스템을 삭제하려면 **cgclear** 명령을 사용합니다.

cgroup에 있는 모든 작업은 계층의 root 노드에 재할당됩니다. 모든 cgroup은 삭제되고 파일 시스템 자체는 시스템에서 마운트 해제됩니다. 따라서 이전에 마운트된 모든 계층을 제거합니다. 마지막으로 cgroup 파일 시스템이 마운트된 디렉토리는 실제로 삭제됩니다.

참고

mount 명령을 사용하여 cgroup을 생성하려면 (**cgconfig** 서비스를 사용하여 생성한 경우와 반대로) **/etc/mtab** 파일에 (마운트된 파일 시스템 테이블)항목이 생성됩니다. 이러한 변경 사항은 **/proc/mounts** 파일에도 반영됩니다. 하지만 **cgclear** 명령 및 기타 다른 **cgconfig** 명령을 사용하여 cgroup을 업로드하는 경우, 변경 사항을 **/etc/mtab** 파일에 반영하지 않고 새로운 정보 만을 **/proc/mounts** 파일에 기록하는 커널 인터페이스가 사용됩니다. 따라서 **cgclear** 명령을 사용하여 cgroup을 업로드한 후에 마운트 해제된 cgroup은 **/etc/mtab** 파일에서 여전히 볼 수 있고 그 결과 **mount** 명령을 실행하면 나타날 수 있습니다. 마운트된 모든 cgroup의 정확한 목록을 확인하려면 **/proc/mounts** 파일을 참조하는 것이 좋습니다.

2.12. 추가 자원

cgroup 명령에 대한 결정적인 문서는 libcgroup 패키지로 제공되는 man 페이지입니다. 섹션 번호는 아래의 man 페이지 목록에 기재되어 있습니다.

libcgroup의 Man 페이지

- **man 1 cgclassify** — **cgclassify** 명령은 하나 이상의 cgroup으로 실행 중인 작업을 이동하는데 사용됩니다.

  man 1 cgclear — **cgclear** 명령은 계층 구조에 있는 모든 cgroup을 삭제하는데 사용됩니다.

  man 5 cgconfig.conf — cgroup은 **cgconfig.conf** 파일에서 정의됩니다.

  man 8 cgconfigparser — **cgconfigparser** 명령은 **cgconfig.conf** 파일을 구문 분석하여 계층을 마운트합니다.

  man 1 cgcreate — **cgcreate** 명령을 계층 구조에 새로운 cgroup을 생성합니다.

  man 1 cgdelete — **cgdelete** 명령은 특정 cgroup을 삭제합니다.

  man 1 cgexec — **cgexec** 명령은 특정 cgroup에 있는 작업을 실행합니다.

  man 1 cgget — **cgget** 명령은 cgroup 매개 변수를 표시합니다.

  man 5 cgred.conf — **cgred.conf**는 **cgred** 서비스의 설정 파일입니다.

man 5 cgrules.conf — **cgrules.conf**에는 특정 cgroup에 속하는 작업이 있는지를 결정하는데 사용되는 규칙이 포함되어 있습니다.

man 8 cgrulesengd — **cgrulesengd** 서비스는 작업을 cgroup에 배분합니다.

man 1 cgset — **cgset** 명령은 cgroup에 대해 매개 변수를 설정합니다.

man 1 lscgroup — **lscgroup** 명령은 계층 구조에 cgroup을 나열합니다.

man 1 lssubsys — **lssubsys** 명령은 특정 서브시스템에 포함된 계층 구조를 나열합니다.

---

[3] **lssubsys** 명령은 libcgroup 패키지에 의해 제공되는 유틸리티 중 하나로 이를 사용하기 위해 libcgroup 을 설치해야 합니다. **lssubsys** 명령을 실행할 수 없을 경우 2장. *컨트롤 그룹 사용하기* 에서 참조하십시오.

3장. 서브시스템 및 조정 가능한 매개 변수

*서브시스템 (Subsystems)*은 cgroup을 인식하는 커널 모듈입니다. 일반적으로 서브시스템은 다른 cgroup에게 다양한 수준의 시스템 자원을 할당할 수 있는 자원 컨트롤러입니다. 하지만 서브시스템은 서로 다른 그룹의 프로세스를 각각 다르게 다뤄야 할 필요가 있을 경우, 커널과 다른 형태의 상호작용을 하도록 프로그램될 수 있습니다. 새로운 서브시스템을 개발하기 위한 *어플리케이션 프로그래밍 인터페이스 (application programming interface)*(API)는 시스템의 **/usr/share/doc/kernel-doc-*kernel-version*/Documentation/cgroups/**에 설치된 (kernel-doc 패키지로 제공) 커널 문서의 **cgroups.txt**에 기재되어 있습니다. cgroup 문서의 최신 버전은 온라인 http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt에서 보실 수 있습니다. 하지만 최신 버전의 문서에 설명된 기능은 사용자의 시스템에 설치된 커널에서 사용할 수 있는 기능과 일치하지 않을 수 있다는 점에 유의하십시오.

cgroup의 서브시스템 매개 변수를 포함하는 *상태 객체(State objects)*는 cgroup의 가상 파일 시스템에서 *가상파일(pseudofiles)*로 표시됩니다. 이러한 가상파일은 셸 명령이나 시스템 호출로 조작할 수 있습니다. 예를 들어, **cpuset.cpus**는 cgroup에 액세스를 허용하는 CPU를 지정하는 가상 파일입니다. 시스템에서 실행되는 웹 서버의 cgroup이 **/cgroup/cpuset/webserver**인 경우 다음 명령을 실행합니다:

```
~]# echo 0,2 > /cgroup/cpuset/webserver/cpuset.cpus
```

위의 명령은 **0,2** 값을 **cpuset.cpus** 가상 파일에 작성하여 **/cgroup/cpuset/webserver/tasks**에 기재된 PID 작업이 시스템의 CPU 0과 CPU 2만을 사용하도록 제한합니다.

3.1. BLKIO

블록 I/O (**blkio**) 서브시스템은 cgroup의 작업을 사용하여 블록 장치의 I/O에 대한 액세스를 제어 및 모니터링합니다. 이러한 가상 파일에 값을 기록하면 액세스 또는 대역폭이 제한되며 이러한 가상 파일에서 값을 읽을 때 I/O 작업에 대한 정보가 제공됩니다.

blkio.weight

기본적으로 cgroup에 제공되는 블록 I/O 액세스의 상대적인 비율 (*가중치*)을 **100**에서 **1000**으로 지정합니다. 이 값은 특정 장치에 대해 **blkio.weight_device** 매개 변수에 의해 덮어쓰기됩니다. 예를 들어, 블록 장치에 액세스하기 위해 기본값 가중치 **500**을 cgroup에 할당하려면 다음 명령을 실행합니다:

```
echo 500 > blkio.weight
```

blkio.weight_device

특정 장치에서 cgroup에 제공되는 I/O 액세스의 상대적 비율 (*가중치*)을 **100**에서 **1000** 이내로 지정합니다. 이 매개 변수 값은 지정된 장치의 **blkio.weight** 매개 변수의 값을 덮어쓰기합니다. 이 값은 *major:minor weight* 형식을 취하며 여기서 *major*와 *minor*는 http://www.kernel.org/doc/Documentation/devices.txt에서 사용 가능한 *Linux 할당 장치* (*Linux 장치 목록*이라고도 알려짐)에서 지정된 장치 유형과 노드 번호입니다. 예를 들어 **/dev/sda**로 액세스하기 위해 cgroup에 **500**의 가중치를 할당하려면 다음 명령을 실행합니다:

```
echo 8:0 500 > blkio.weight_device
```

Linux 할당 장치 표기에서 **8:0**은 **/dev/sda**를 나타냅니다.

blkio.time

cgroup이 특정 장치로의 I/O 액세스 시간을 보고합니다. 항목은 *major*, *minor*, *time*의 3 개의 필드로 구성됩니다. *Major* 및 *minor*는 *Linux 할당 장치에* 지정된 장치 유형과 노드 번호이며 *time*은 밀리 초 (ms) 단위의 시간입니다.

blkio.sectors

cgroup에 의해 특정 장치 간에 전송되는 섹터 수를 보고합니다. 항목은 *major*, *minor*, *sectors*의 3 개 필드로 구성됩니다. *Major*와 *minor*는 *Linux 할당 장치*에 지정된 장치 유형과 노드 번호이며 *sectors*는 디스크 섹터 수입니다.

blkio.io_service_bytes

cgroup에 의해 특정 장치 간에 전송되는 바이트 수를 보고합니다. 항목은 *major*, *minor*, *operation*, *bytes*의 4개 필드로 구성됩니다. *Major*와 *minor*는 *Linux 할당 장치*에 지정된 장치 유형과 노드 번호입니다. *operation*은 작업 유형 **read**, **write**, **sync**, **async**)을 나타내며, *bytes*는 전송된 바이트 수를 나타냅니다.

blkio.io_serviced

특정 장치에서 cgroup에 의해 실행된 I/O 작업의 수를 보고합니다. 항목은 *major*, *minor*, *operation*, *bytes*의 4 개 필드로 구성됩니다. *Major*와 *minor*는 *Linux 할당 장치*에서 지정된 장치 유형 및 노드 수입니다. *operation*은 작업 유형 (**read**, **write**, **sync**, **async**)을 나타내며 *number*는 작업 수를 나타냅니다.

blkio.io_service_time

cgroup에 의한 특정 장치에서 발생하는 I/O 작업 요청의 발송에서 완료 까지의 총 시간을 보고합니다. 항목은 *major*, *minor*, *operation*, *bytes*의 네 개의 필드로 구성됩니다. *Major*와 *minor*는 *Linux 할당 장치*에 지정된 장치 유형과 노드 번호입니다. *operation*은 작업 유형 (**read**, **write**, **sync**, **async**)을 나타내고 *time*은 나노초 (ns) 단위로 표시됩니다. 시간은 큰 단위가 아닌 나노초 단위로 보고되기 때문에 이러한 보고서는 솔리드 스테이트 장치의 경우에도 의미가 있습니다.

blkio.io_wait_time

스케줄러 대기열에서 서비스를 기다리는 데 소요된 cgroup에 의한 특정 장치의 총 I/O 작업 시간을 보고합니다. 이 보고서를 분석할 때 다음 사항에 유의하십시오:

- 보고되는 시간은 cgroup 자체가 I/O 작업을 기다리는데 소요된 시간이 아니라 cgroup 의 모든 I/O 작업의 누계이기 때문에 총 경과 시간 보다 길 수 있습니다. 그룹 전체로 소요된 대기 시간을 확인하려면 **blkio.group_wait_time**을 사용합니다.

- 장치가 **queue_depth** > 1을 갖는 경우, 보고되는 시간은 장치가 요청을 다시 정렬하는 동안 소요된 대기 시간이 아닌 요청이 장치에 발송될 때 까지의 시간 만을 포함합니다.

항목은 *major*, *minor*, *operation*, *bytes*의 네 개의 필드로 구성됩니다. *Major*와 *minor*는 *Linux 할당 장치*에 지정된 장치 유형과 노드 번호입니다. *operation*은 작업 유형 (**read**, **write**, **sync**, **async**)을 나타내고 *time*은 나노초 (ns) 단위로 표시됩니다. 시간은 큰 단위가 아닌 나노초 단위로 보고되기 때문에 이러한 보고서는 솔리드 스테이트 장치의 경우에도 의미가 있습니다.

blkio.io_merged

cgroup에 의해 I/O 작업에 대한 요청으로 병합된 BIOS 요청 수를 보고합니다. 항목은 *number*와 *operation*의 두 개의 필드로 구성됩니다. *Number*는 요청 수 이고 *operation*은 작업 유형 (**read**, **write**, **sync**, **async**)을 나타냅니다.

blkio.io_queued

cgroup에 의한 I/O 작업 대기열의 요청 수를 보고합니다. 항목은 *number*와 *operation*의 두 개의 필드로 구성됩니다. *Number*는 요청 수이며, *operation*은 작업 유형 (**read**, **write**, **sync**, **async**)을 나타냅니다.

blkio.avg_queue_size

전체 그룹 존재 시간에 걸쳐 cgroup에 의한 I/O 작업의 평균 대기열 크기를 보고합니다. 대기열 크기는 cgroup의 대기열이 타임 슬라이스를 얻을 때마다 샘플링됩니다. 이 보고서는 시스템에서 **CONFIG_DEBUG_BLK_CGROUP=y**가 설정되어 있는 경우에만 사용할 수 있다는 점에 유의하십시오.

blkio.group_wait_time

cgroup 하나의 대기열의 타임 슬라이스에 대해 cgroup이 소요한 대기 시간의 합계를 (나노 초단위 - ns) 보고합니다. 보고서는 cgroup의 대기열이 타임 슬라이스를 얻을 때 마다 업데이트되므로 cgroup이 타임 슬라이스를 기다리는 동안 가상 파일을 읽는 경우 보고서에는 현재 대기열에 있는 작업을 기다리는 데 소비한 시간은 포함되지 않습니다. 이 보고서는 시스템에서 **CONFIG_DEBUG_BLK_CGROUP=y**가 설정되어 있는 경우에만 사용할 수 있는 점에 유의하십시오.

blkio.empty_time

보류 중인 요청 없이 cgroup이 소요한 총 시간을 (나노 초단위 - ns) 보고합니다. 보고서는 보류 중이 요청이 cgroup 대기열에 들어갈 때 마다 업데이트되므로 cgroup이 보류 중인 요청이 없이 가상 파일을 읽을 경우, 현재 빈 상태에서 소비한 시간은 보고서에 포함되지 않게 됩니다. 이 보고서는 시스템에서 **CONFIG_DEBUG_BLK_CGROUP=y**가 설정된 경우에만 사용 가능하다는 점에 유의하십시오.

blkio.idle_time

이미 대기열 또는 다른 그룹에 있는 요청 보다 상위의 요청을 예상하여 cgroup에 대해 스케줄러가 유휴 상태에 소비된 총 시간을 (나노초 단위 — ns) 보고합니다. 보고서는 그룹이 더이상 유휴 상태에 있지 않을 때 마다 업데이트되므로 cgroup이 유휴 상태에 있는 동안 가상 파일을 읽을 경우, 현재 유휴 상태에서 소요된 시간은 보고서에 포함되지 않습니다. 이 보고서는 시스템에서 **CONFIG_DEBUG_BLK_CGROUP=y**가 설정되어 있는 경우에만 사용할 수 있다는 점에 유의하십시오.

blkio.dequeue

특정 장치의 대기열에서 삭제된 cgroup에 의해 요청된 I/O 작업의 수를 보고합니다. 항목은 *major*, *minor*, *number*의 3개의 필드로 구성됩니다. *Major* 및 *minor*는 *Linux 할당 장치*에 지정된 장치 유형 및 노드 번호입니다. *number*는 그룹이 대기열에서 삭제 요청된 횟수입니다. 이 보고서는 시스템에서 **CONFIG_DEBUG_BLK_CGROUP=y**가 설정되어 있는 경우에만 사용할 수 있다는 점에 유의하십시오.

blkio.reset_stats

기타 다른 가상 파일에 기록되는 통계를 재설정합니다. 이 cgroup에 대한 통계를 다시 설정하려면 이 파일에 정수를 씁니다.

3.2. CPU

cpu 서브시스템은 cgroup에 대해 CPU 액세스를 스케줄링합니다. CPU 자원에 대한 액세스는 다음 매개변수에 따라서 관리되며, cgroup 가상 파일 시스템에서 각각의 CPU는 별도의 *가상파일 (pseudofile)*로 다뤄집니다.

cpu.shares

cgroup에 있는 작업에서 사용 가능한 상대적인 CPU 시간 점유 비율을 지정하는 정수를 포함합니다. 예를 들어 두 cgroup에 속한 작업의 **cpu.shares**가 **1**로 설정되어 있다면 그 두 작업은 같은 CPU 시간을 배정받습니다. 하지만, 어떤 cgroup의 작업의 **cpu.shares**가 **2**로 설정되어 있다면, 그 작업은 **cpu.shares**가 **1**로 설정된 작업보다 두 배의 CPU 시간을 배정받습니다.

cpu.rt_runtime_us

cgroup 내의 작업이 CPU 자원을 최대한으로 연속적으로 액세스할 수 있는 시간을 마이크로초 (µs, 항목 이름에는 "us"라고 되어있음) 단위로 지정합니다. 이 값을 지정하면 한 cgroup 내의 작업이 CPU 시간을 독점하는 것을 방지할 수 있습니다. 만약 어떤 cgroup의 작업이 CPU 자원을 5초 중에 4초간 액세스 할 수 있어야 한다면 **cpu.rt_runtime_us**를 **4000000**로 설정하고, **cpu.rt_period_us**를**5000000**로 설정합니다.

cpu.rt_period_us

cgroup에 대해 CPU 자원이 정기적으로 재할당되야 하는 주기를 마이크로초(µs, 항목 이름에는 "us"라고 되어있음) 단위로 지정합니다. 만약 어떤 cgroup의 작업이 CPU 자원을 5초 중에 4초간 액세스 할 수 있어야 한다면 **cpu.rt_runtime_us**를 **4000000**로 설정하고, **cpu.rt_period_us**을 **5000000**로 설정하십시오.

3.3. CPUACCT

CPU Accounting (**cpuacct**) 서브시스템은 자식 그룹에 있는 작업을 포함하여 cgroup에 있는 작업에 사용되는 CPU 자원에 대한 보고서를 자동으로 생성합니다. 세가지 보고서를 사용할 수 있습니다:

cpuacct.stat

사용자 모드와 시스템(커널) 모드에서 이 cgroup의 작업과 자식 작업이 사용한 CPU 사이클 수 (**USER_HZ**로 시스템에 정의된 단위를 사용)를 보고합니다.

cpuacct.usage

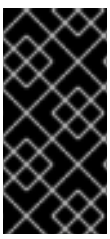
cgroup의 모든 작업 (계층구조에서 하위계층에 속하는 작업도 포함)이 사용한 전체 CPU 시간 (나노초 단위)을 보고합니다.

cpuacct.usage_percpu

이 cgroup 모든 작업 (계층구조에서 하위계층의 작업을 포함)은 각각의 CPU에서 소모한 CPU 시간 (단위:나노초)을 보고합니다 .

3.4. CPUSET

cpuset 서브시스템은 개별 CPU와 메모리 노드를 cgroup에 할당합니다. 각각의 cpuset은 cgroup 가상 파일 시스템 내의 별도의 *가상 파일 (pseudofile)*에 각각 다음 매개변수를 통해 지정될 수 있습니다.



중요

일부 서브시스템은 이러한 서브시스템을 사용하는 cgroup으로 작업을 이동하기 전 설정해야 하는 필수 매개 변수를 갖습니다. 예를 들어, **cpuset** 서브시스템을 사용하는 cgroup에 작업을 이동하기 전 **cpuset.cpus** 및 **cpuset.mems** 매개 변수를 반드시 정의해야 합니다.

cpuset.cpus (필수)

cgroup에 속한 작업에대해 액세스가 허용된 CPU를 지정합니다. 이것은 ASCII 형태로 된 콤마로 구분된 목록입니다. 범위를 표현할 때는 대시("**-**")를 사용합니다. 예:

```
0-2,16
```

위와 같은 것은 CPU 0,1,2와 16을 표현합니다.

cpuset.mems (필수)

cgroup에 있는 작업에 대해 액세스가 허용된 메모리 노드를 지정합니다. 이는 ASCII 형식으로된 콤마로 분리된 목록입니다. 범위를 표현할 때는 대시 ("**-**")를 사용합니다. 예:

```
0-2,16
```

위와 같은 것은 메모리 노드 0,1,2와 16을 표현합니다.

cpuset.memory_migrate

cpuset.mems의 값이 변경되면 메모리의 페이지가 새로운 노드로 이동해야 하는지를 지정하는 플래그(**0**또는 **1**)를 포함합니다. 디폴트로 메모리 이동은 금지(**0**)되어 있으며, 페이지는 최초 할당된 메모리 노드에 남아있게 됩니다. 심지어 **cpuset.mems**에 새롭게 지정된 노드가 최초 할당된 노드와 달라지는 경우에도 그렇습니다. 만약 활성화(**1**)되면 시스템은 페이지를 **cpuset.mems**에 지정된 새로운 매개변수에 맞춰, 가능하면 상대적인 위치도 유지하면서, 이동시킬 것입니다. 예를 들어 **cpuset.mems**에 의해 최초 두번째 노드에 있도록 지정된 페이지들은 **cpuset.mems**에 의해 새로 지정된 두번째 노드가 사용 가능하다면 그 노드에 할당될 것입니다.

cpuset.cpu_exclusive

부모, 자식, cpuset 이외에 다른 cpuset이 이 cpuset에 의해 지정된 CPU를 공유해야 할지 여부를 지정하는 플래그 (**0**또는 **1**)를 포함합니다. 디폴트(**0**)로 CPU는 한 cpuset에만 배타적으로 할당되지 않습니다.

cpuset.mem_exclusive

다른 cpuset들이 이 cpuset에 의해 지정된 메모리 노드들을 공유할 수 있는지를 지정하는 플래그(**0**또는 **1**)를 포함합니다. 디폴트(**0**)로 메모리 노드들은 한 cpuset에만 배타적으로 할당되지 않습니다. 특정 cpuset에서만 메모리 노드를 사용하도록 독점권을 부여하는 것은(**1**) **cpuset.mem_hardwall**로 메모리 격벽(memory hardwall)을 설정하는 것과 기능적으로 동일한 것입니다.

cpuset.mem_hardwall

커널의 메모리 페이지와 버퍼 데이터 할당이 이 cpuset에 지정된 메모리 노드들로만 제한되야 하는지 여부를 지정하는 플래그(**0**또는 **1**)를 포함합니다. 디폴트(**0**)로, 페이지와 버퍼 데이터는 여러 사용자에 속하는 프로세스들 사이에서 공유됩니다. 격벽을 설정(**1**)시, 각각의 태스크의 메모리 할당은 별도로 유지됩니다.

cpuset.memory_pressure

이 cpuset에 속한 프로세스들에 의한 *메모리 압력(memory pressure)*의 현재 평균값을 포함하는 읽기 전용 파일입니다. 이 가상파일(pseudofile)의 값은 **cpuset.memory_pressure_enabled**가 지정된 경우 자동으로 업데이트됩니다. 그렇지 않은 경우 **0** 값을 포함합니다.

cpuset.memory_pressure_enabled

이 cgroup에 있는 프로세스에 의해 생성된 *메모리 압력(memory pressure)*을 시스템이 계산해야 할 지를 결정하는 플래그(**0**,**1**)를 포함합니다. 계산된 값은 **cpuset.memory_pressure**에 지정된 파일에 기록되며, 그 값은 초당 메모리 재할당 요청 횟수를 1000으로 곱한 정수값으로 표현되는, 프로세스가 사용중인 메모리를 해제하려는 경향을 표시합니다.

cpuset.memory_spread_page

파일시스템 버퍼가 이 cpuset에 할당된 메모리 노드 사이에 균등하게 배분되어야 할지를 지정하는 플래그(**0**,**1**)를 포함합니다. 디폴트(**0**)로 균등화를 위한 노력을 시도하지 않으며, 버퍼는 그 버퍼를 생성

한 프로세스가 실행중인 것과 같은 노드에 할당됩니다.

cpuset.memory_spread_slab

파일 입출력을 위한 커널 슬랩(slab) 캐시가 cpuset 전체에 걸쳐 균일하게 배분되어야 할지를 지정하는 플래그(**0**,**1**)를 포함합니다. 디폴트(**0**)로, 커널 슬랩 캐시를 균일하게 하려는 노력을 하지 않습니다. 슬랩 캐시는 그것을 생성한 프로세스가 실행중인 메모리 노드와 동일한 것에 위치하게 됩니다.

cpuset.sched_load_balance

커널이 cpuset내의 CPU 사이에 부하를 균일하게 배분할지를 지정하는 플래그(**0**,**1**)를 포함합니다. 디폴트(**1**)로, 커널은 과부화인 CPU에서 부하가 더 적은 CPU로 프로세스를 옮겨서 부하를 균일화합니다.

하지만 부모 cgroup에서 로드 밸런싱이 활성화되어 있는 경우 로드 밸런싱은 높은 수준에서 이미 실행되고 있으므로 cgroup에서 이 플래그를 설정하는 것은 영향을 미치지 않습니다. 따라서 cgroup에서 로드 밸런싱을 비활성화하려면 계층의 각 부모 cgroup에 있는 로드 밸런싱을 비활성화해야 합니다. 이 경우 cgroup의 형제 관계에 있는 그룹에 대해서도 로드 밸런싱을 활성화해야 할 지에 대한 여부도 고려해야 합니다.

cpuset.sched_relax_domain_level

-1과 어떤 작은 양수값 사이의 정수를 포함하며, 이 숫자는 커널이 부하를 균등화하기 위해 사용해야 하는 CPU의 갯수를 표현합니다. 만약 **cpuset.sched_load_balance**가 비활성화 되어 있다면 이 값은 의미가 없습니다.

지정된 값의 효과는 시스템 아키텍쳐에 따라 달라지지만, 다음과 같은 값을 일반적으로 사용합니다:

cpuset.sched_relax_domain_level의 값

값	효과
-1	부하 균등화에 시스템 디폴트값을 사용함
0	직접적인 부하 균일화를 수행하지 않음; 주기적으로만 부하를 균등화함
1	동일한 코어상의 쓰레드간 부하를 직접 균등화
2	동일한 패키지 내의 코어간 부하를 직접 균등화
3	동일한 노드/블레이드 상의 CPU간의 부하를 직접 균등화
4	NUMA(비균등 메모리 액세스,non-uniform memory access) 아키텍쳐상에서 여러 CPU간에 직접적으로 부하 균등화
5	NUMA 아키텍쳐상에서 모든 CPU간의 부하를 직접적으로 균등화

3.5. DEVICES

devices 서브시스템은 cgroup의 작업을 사용하여 장치에 대한 액세스를 허용하거나 거부합니다.

중요

Red Hat Enterprise Linux 6에서 Device Whitelist (**devices**) 서브시스템은 기술 프리뷰로 간주됩니다.

기술 프리뷰 기능은 현재 Red Hat Enterprise Linux 6 서브스크립션 서비스에서는 지원되지 않으며, 기능적으로 완전하지 않고 프로덕션 환경에서 사용하기에는 적합하지 않을 수 있습니다. 하지만 Red Hat은 고객의 편의를 위해 이러한 기능을 운영 체제에서 폭넓게 공개하고 있습니다. 이러한 기능은 비프로덕션 환경에서 유용하게 사용될 수 있으며 완전 지원되기 전에 기술 프리뷰 기능에 대한 피드백 또는 기능적 제안을 해주시기 바랍니다.

devices.allow

cgroup 내의 작업에 대한 액세스가 허용되야 하는 장치를 지정합니다. 각각의 항목은 4개의 필드가 있습니다: *type*, *major*, *minor*, *access*. *type*, *major*, *minor* 필드에 사용된 값은 *Linux 할당 장치*에 지정된 장치 유형 및 노드 번호에 해당합니다. (http://www.kernel.org/doc/Documentation/devices.txt에서는 *Linux 장치 목록*이라고 부릅니다)

type

*type*은 다음 세가지 값 중 하나일 수 있습니다:

- **a**는 *문자 장치(character devices)*와 *블럭 장치(block devices)* 모두에 대응됩니다
- **b**는 블럭 장치를 지정합니다
- **c**는 문자 장치를 지정합니다

major , minor

*major*와 *minor*는 *Linux 할당 장치*에 지정된 장치 노드 번호입니다. 메이저와 마이너 번호는 콜론으로 구분됩니다. 예를 들어 **8**은 SCSI 디스크 드라이브를 표시하는 메이저 번호이고, **1**은 첫번째 SCSI 디스크 드라이브의 첫번째 파티션을 지정하는 마이너 번호입니다; 따라서 **8:1**은 전체적으로 파일시스템에서 **/dev/sda1**에 있는 파티션을 지정합니다.

*****는 모든 메이저, 마이너 노드에 대응합니다. 예: **9:***(모든 RAID 장치), ***:***(모든 장치)

access

*access*는 다음 문자들이 하나 이상 포함된 문자열입니다:

- **r**은 태스크가 지정된 장치에서 읽는것을 허용합니다
- **w**는 태스크가 지정된 장치에 쓰는것을 허용합니다
- **m**는 아직 존재하지 않는 장치를 만드는 것을 허용합니다

예를 들어 *access*가 **r**로 지정되면, 태스크는 해당 장치에서 읽을 수만 있습니다. 만약 *access*가 **rw**로 지정되었다면, 태스크는 그 장치에 읽기와 쓰기가 모두 가능합니다.

devices.deny

cgroup의 작업에 액세스할 수 없는 장치를 지정합니다. 항목의 구문은 **devices.allow**와 동일합니다.

devices.list

이 cgroup에 있는 작업에 대해 액세스 제어가 설정된 장치를 보고합니다.

3.6. FREEZER

freezer 서브시스템은 cgroup의 작업을 일시 중지하거나 재개합니다.

freezer.state

freezer.state는 3가지 값을 가질 수 있습니다:

- **FROZEN** — cgroup의 작업이 일시 중단 상태에 있습니다.
- **FREEZING** — 시스템이 cgroup의 작업을 일시 중지하는 중입니다.
- **THAWED** — cgroup의 작업을 재개하고 있습니다.

특정 프로세스를 일시 중지하려면 다음을 실행합니다:

1. **freezer** 서브시스템이 연결된 계층에 있는 cgroup으로 프로세스를 이동합니다.
2. 특정 cgroup을 중지하여 그 안에 포함된 프로세스를 일시 중지시킵니다.

일시 중지 (중지)한 cgroup으로 프로세스를 이동할 수 없습니다.

FROZEN과 **THAWED** 값은 **freezer.state**에 입력할 수 있지만, **FREEZING**는 입력할 수는 없고, 읽을수만 있다는 것에 유의하십시오.

3.7. MEMORY

memory 서브시스템은 cgroup의 작업에서 사용되는 메모리 리소스에 대한 보고서를 자동으로 생성하고 다른 작업을 사용하여 메모리 사용의 제한을 설정합니다:

memory.stat

다음 표에서 볼 수 있듯이 광범위한 메모리 통계를 보고합니다:

표 3.1. 메모리 통계가 보고하는 값들

통계	설명
cache	**tmpfs** (**shmem**)를 포함하는 페이지 캐시 (바이트 단위)
rss	**tmpfs** (**shmem**)를 *포함하지 않는* 익명 스왑 캐시 (바이트 단위)
mapped_file	**tmpfs** (**shmem**)를 포함한 메모리 맵핑된 파일의 크기 (바이트 단위)
pgpgin	메모리에 페이지된 페이지 갯수

통계	설명
pgpgout	메모리에서 페이지아웃된 페이지 갯수
swap	스왑 사용량 (바이트 단위)
active_anon	**tmpfs** (**shmem**)를 포함한 활성화된 최근 최소 사용(LRU,least-recently-used) 목록에 있는 무명/스왑 캐시 (바이트 단위)
inactive_anon	**tmpfs** (**shmem**)를 포함한 비활성 LRU 목록에서 익명의 스왑 캐시 (바이트 단위)
active_file	활성화된 LRU 목록에 있는 파일 기반 메모리(바이트 단위)
inactive_file	비활성화된 LRU 목록에 있는 파일기반 메모리 (바이트 단위)
unevictable	재활용할 수 없는 메모리 (바이트 단위)
hierarchical_memory_limit	**memory** cgroup을 포함하는 계층의 메모리 제한 (바이트 단위)
hierarchical_memsw_limit	**memory** cgroup을 포함하는 계층의 메모리와 스왑 제한 (바이트 단위)

또한 **hierarchical_memory_limit** 및 **hierarchical_memsw_limit** 이외의 파일은 각각 cgroup 뿐만 아니라 모든 자식 그룹에도 보고하는 **total_** 접두사를 갖습니다. 예를 들어, **swap**은 cgroup에 의한 스왑 사용량을 보고하고 **total_swap**은 cgroup과 자식 그룹에 의한 총 스왑 사용량을 보고합니다.

memory.stat에 의해 보고된 값을 구문 분석할 때 다양한 통계가 서로 관련되어 있다는 점에 유의하십시오:

- **active_anon** + **inactive_anon** = 익명 메모리 + **tmpfs** 파일 캐시 + 스왑 캐시

  따라서 **active_anon** + **inactive_anon** ≠ **rss**가 됩니다. 이는 **rss**에는 **tmpfs**가 포함되지 않기 때문입니다.

- **active_file** + **inactive_file** = cache - size of **tmpfs**

memory.usage_in_bytes

cgroup에서 프로세스에 의해 현재 사용되는 전체 메모리 사용량을 보고합니다 (바이트 단위).

memory.memsw.usage_in_bytes

cgroup에서 프로세스에 의해 사용되는 현재 메모리 사용량과 스왑 영역 사용량의 합계를 보고합니다 (바이트 단위).

memory.max_usage_in_bytes

cgroup에서 프로세스에 의한 최대 메모리 사용량을 보고합니다 (바이트단위).

memory.memsw.max_usage_in_bytes

cgroup에서 프로세스에 의해 사용된 스왑 영역 및 최대 메모리 사용량을 보고합니다 (바이트 단위).

memory.limit_in_bytes

최대 사용자 메모리양 (파일 캐시 포함)을 설정합니다. 단위가 지정되어 있지 않으면 값은 바이트 단위로 해석됩니다. 하지만, 더 큰 단위를 숫자 뒤에 사용할 수 도 있습니다. **k**,**K**는 킬로바이트, **m**,**M**는 메가바이트, **g**,**G**는 기가바이트를 의미합니다.

root cgroup을 제한하기 위해 **memory.limit_in_bytes**는 사용할 수 없습니다. 계층에 있는 하위 그룹에만 값을 적용할 수 있습니다.

memory.limit_in_bytes에 **-1**을 작성하여 기존 제한을 제거합니다.

memory.memsw.limit_in_bytes

총 메모리와 스왑 사용량의 최대 크기를 지정합니다. 단위를 지정하지 않으면 값은 바이트 단위로 해석됩니다. 하지만 더 큰 단위를 숫자 뒤에 사용할 수 있습니다. **k**,**K**는 킬로바이트, **m**,**M**는 메가바이트, **g**,**G**은 기가바이트를 의미합니다.

root cgroup을 제한하기 위해 **memory.memsw.limit_in_bytes**는 사용할 수 없습니다. 계층에 있는 하위 그룹에만 값을 적용할 수 있습니다.

memory.memsw.limit_in_bytes에 **-1**로 작성하여 기존 제한을 제거합니다.

memory.failcnt

memory.limit_in_bytes로 설정된 메모리 제한에 도달한 횟수를 보고합니다.

memory.memsw.failcnt

memory.memsw.limit_in_bytes에 설정된 메모리와 스왑 공간의 합계가 상한에 도달한 횟수를 보고합니다.

memory.force_empty

0으로 설정하면, cgroup의 작업에서 사용되는 모든 페이지의 메모리를 비웁니다. 이 인터페이스는 cgroup에 작업이 없는 경우에만 사용할 수 있습니다. 메모리를 해제할 수 없을 경우, 가능하면 부모 cgroup으로 이동됩니다. cgroup을 삭제하기 전 **memory.force_empty**를 사용하여 사용되지 않는 페이지 캐시가 부모 cgroup에 이동되지 않게 합니다.

memory.swappiness

이 cgroup에 속한 작업이 사용중인 프로세스 메모리를 커널이 페이지 캐시에서 페이지를 재활용하지 하는 대신에 스왑 아웃하는 경향을 지정합니다. 이 값은 **/proc/sys/vm/swappiness**에 시스템 전체로 지정된 경향과 정확히 같은 방식으로 계산되는 경향 값입니다. 기본 값은 **60**입니다. 이 값보다 더 낮은 값으로 지정하면 커널이 프로세스 메모리를 스왑아웃 하는 경향을 감소시키고, **60** 보다 더 높은 값으로 지정하면 커널이 프로세스 메모리를 스왑 아웃하는 경향을 증가시킵니다. **100**보다 더 큰 값을 지정하면, 이 cgroup에 속한 프로세스의 주소 공간에 있는 페이지들을 커널이 스왑 아웃하는 것을 허용합니다.

0의 값으로 설정하면 프로세스 메모리가 스왑 아웃되는 것을 막을 수 없다는 점에 유의하십시오. 글로벌 가상 메모리 관리 논리는 cgroup 값을 읽지 않기 때문에 시스템 메모리가 부족한 경우 여전히 스왑 아웃이 발생할 수 있습니다. 페이지를 완전히 잠금하려면 cgroup 대신 **mlock()**을 사용합니다.

다음과 같은 그룹의 swappiness는 변경할 수 없습니다:

- **/proc/sys/vm/swappiness**에 설정된 swappiness를 사용하는 root cgroup
- 아래에 자식 그룹을 갖는 cgroup

memory.use_hierarchy

cgroup의 계층구조 전체에 대해 메모리 사용량이 계산되어야 할지를 지정하는 플래그(**0** 또는 **1**)가 포함됩니다. 만약 활성화되면(**1**로 지정), 메모리 서브시스템은 메모리 제한을 넘어간 프로세스와 그 자식 프로세스로부터 메모리를 재활용하게 됩니다. 디폴트값(**0**)에서 서브시스템은 작업의 자식에서 메모리를 재활용하지 않습니다.

3.8. NET_CLS

net_cls 서브시스템은 네트워크 패킷을 클래스 식별자(classid)로 태그하여 Linux 트래픽 컨트롤러 (**tc**)가 특정 cgroup에서 발생하는 패킷들을 식별할 수 있게 합니다. 트래픽 컨트롤러는 서로 다른 cgroup에서 발생하는 패킷에 대해 다른 우선 순위를 할당하도록 설정할 수 있습니다.

net_cls.classid

net_cls.classid에는 트래픽 컨트롤 *핸들(handle)*을 표시하는 16진수 값이 포함되어 있습니다. 예를 들어 **0x100001**은 iproute2에 의해 사용되는 형식에서 핸들을 **10:1**라고 쓰기도 합니다.

이러한 핸들 형식은 **0x*AAAABBBB***입니다. 여기서 *AAAA*는 16 진수의 주 번호이고 *BBBB*는 16 진수의 부 번호입니다. 또한 선행 0을 생략할 수 있으며 **0x10001**은 **0x00010001**과 같고 **1:1**을 나타냅니다.

tc의 매뉴얼 페이지(man page)를 참조해서 트래픽 컨트롤러를 **net_cls**가 네트워크 패킷에 추가하는 핸들을 활용하도록 설정하는 방법을 배우십시오.

3.9. NS

ns 서브시스템은 프로세스를 다른 *네임 스페이스 (namespaces)*로 그룹화하는 방법을 제공합니다. 특정 네임 스페이스 내에서 프로세스 간의 상호 작용이 가능하지만 다른 네임 스페이스에서 실행 중인 프로세스에서 분리됩니다. 이러한 분리된 네임 스페이스는 운영 체제 수준의 가상화에 사용되는 경우 *컨테이너 (container)*라고도 부릅니다.

3.10. 추가 자원

특정 서브시스템의 커널 문서

다음 파일은 **/usr/share/doc/kernel-doc-<*kernel_version*>/Documentation/cgroups/** 디렉토리 아래에 있습니다 (kernel-doc 패키지로 제공됨).

- **blkio** 서브시스템 — **blkio-controller.txt**
- **cpuacct** 서브시스템 — **cpuacct.txt**
- **cpuset** 서브시스템 — **cpusets.txt**
- **devices** 서브시스템 — **devices.txt**
- **freezer** 서브시스템 — **freezer-subsystem.txt**
- **memory** 서브시스템 — **memory.txt**

부록 A. 개정 내역

고침 1-12.400 **2013-10-31** **Rüdiger Landmann**
Rebuild with publican 4.0.0

고침 1-12 **2012-07-18** **Anthony Towns**
Rebuild for Publican 3.0

고침 1.0-5 **Thu May 19 2011** **Martin Prpič**
Red Hat Enterprise Linux 6.1 *자원 관리 가이드*의 GA 릴리즈

고침 1.0-4 **Tue Mar 1 2011** **Martin Prpič**
다수의 예시를 수정 — BZ#667623, BZ#667676, BZ#667699
cgclear 명령의 명확화 — BZ#577101
lssubsystem 명령의 명확화 — BZ#678517
프로세스 중지 — BZ#677548

고침 1.0-3 **Wed Nov 17 2010** **Rüdiger Landmann**
리마운트 (remount) 예시를 수정 — BZ#612805

고침 1.0-2 **Thu Nov 11 2010** **Rüdiger Landmann**
사전 릴리즈 피드백 지시 사항을 삭제

고침 1.0-1 **Wed Nov 10 2010** **Rüdiger Landmann**
QE에서 수정 — BZ#581702 and BZ#612805

고침 1.0-0 **Tue Nov 9 2010** **Rüdiger Landmann**
GA의 기능 구현 완료 버전