Red Hat Enterprise Linux 6

Guide de gestion des ressources

Gestion des ressources système sur Red Hat Enterprise Linux 6

Édition 1

Last Updated: 2017-10-12

Red Hat Enterprise Linux 6 Guide de gestion des ressources

Gestion des ressources système sur Red Hat Enterprise Linux 6
Édition 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

Notice légale

Résumé

Gestion des ressources système sur Red Hat Enterprise Linux 6

Table des matières

CHAPITRE 1. INTRODUCTION AUX GROUPES DE CONTRÔLE

Red Hat Enterprise Linux 6 offre une nouvelle fonctionnalité du noyau : les *groupes de contrôle* (de l'anglais, « Control Groups »), aussi appelés *cgroups* dans ce guide. Les groupes de contrôle permettent d'allouer des ressources — telles que le temps processeur, la mémoire système, la bande réseau, ou une combinaison de ces ressources — parmi des groupes de tâches (processus) définis par l'utilisateur et exécutés sur un système. Vous pouvez surveiller les groupes de contrôle que vous avez configuré, leur refuser d'accéder à certaines ressources et même les reconfigurer dynamiquement sur un système en cours d'exécution. Le service **cgconfig** (« *control group config* ») peut être configuré pour être lancé lors du démarrage et rétablir vos groupes de contrôle prédéfinis, les rendant ainsi persistants lors de multiples démarrages.

En utilisant les groupes de contrôle, les administrateurs système obtiennent un contrôle précis sur l'allocation, la priorisation, l'interdication, la gestion et la surveillance des ressources système. Les ressources matérielles peuvent être intelligemment divisées entre les tâches et les utilisateurs, améliorant ainsi l'efficacité générale.

1.1. ORGANISATION DES GROUPES DE CONTRÔLE

Les groupes de contrôle sont organisés de manière hiérarchique, comme des processus, et les groupes de contrôle enfants héritent de certains des attributs de leurs parents. Il existe cependant des différences entre les deux modèles.

Modèle du processus Linux

Tous les processus sur un système Linux sont des processus enfants d'un parent commun : le processus **init**, qui est exécutépar le noyau lors du démarrage et démarre d'autres processus (qui pourraient démarrer des processus enfants à leur tour). Comme tous les processus proviennent d'un seul parent, le modèle du processus Linux est consiste d'une hiérarchie ou arborescence unique.

En outre, tous les processus Linux, à l'exception de **init**, héritent de l'environnement (comme la variable PATH)[1] et de certains autres attributs (comme les descripteurs d'ouverture de fichiers) de leur processus parent.

Modèle des groupes de contrôle

Les groupes de contrôle sont similaires aux processus car :

- ils sont organisés hiérarchiquement
- les groupes de contrôle enfants héritent de certains attributs de leur groupe de contrôle parent.

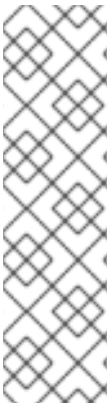
La différence fondamentale est que de nombreuses et diverses hiérarchies de groupes de contrôle peuvent exister simultanément sur un système. Le modèle du processus Linux est une unique arborescence de processus, et le modèle des groupes de contrôle est une ou plusieurs arborescences de tâches (ou de processus) séparées et déconnectées.

De multiples hiérarchies séparées de groupes de contrôle sont nécessaires car chaque hiérarchie est attachée à *un ou plusieurs sous-système*. Un sous-système[2] représente une seule ressource, comme le temps processeur ou la mémoire. Red Hat Enterprise Linux 6 fournit neuf sous-systèmes de groupes de contrôle, répertoriés ci-dessous par nom et par fonction.

Sous-systèmes disponibles dans Red Hat Enterprise Linux

- **blkio** — ce sous-système établit des limites sur l'accès des entrées/sorties à partir et depuis des périphériques blocs tels que des lecteurs physiques (disques dur, disques SSD, lecteurs USB, etc.).

- **cpu** — ce sous-système utilise le planificateur pour fournir aux tâches des groupes de contrôle accès au CPU.
- **cpuacct** — ce sous-système génère des rapports automatiques sur les ressources CPU utilisées par les tâches dans un groupe de contrôle.
- **cpuset** — ce sous-système assigne des CPU individuels (sur un système multicoeur) et des noeuds de mémoire à des tâches dans un groupe de contrôle.
- **devices** — ce sous-système autorise ou refuse l'accès des tâches aux périphériques dans un groupe de contrôle.
- **freezer** — ce sous-système suspend ou réactive les tâches dans un groupe de contrôle.
- **memory** — ce sous-système établit les limites d'utilisation de la mémoire par les tâches d'un groupe de contrôle et génère des rapports automatiques sur les ressources mémoire utilisées par ces tâches.
- **net_cls** — ce sous-système repère les paquets réseau avec un identifiant de classe (classid) qui permet au contrôleur de trafic Linux (**tc**) d'identifier les paquets provenant d'une tâche particulière d'un groupe de contrôle.
- **ns** — le sous-système *namespace*.



NOTE

Vous pouvez rencontrer le terme *contrôleur de ressources* ou plus simplement *contrôleur* dans la littérature concernant les groupes de contrôle, comme dans les pages man ou dans la documentation du noyau. Ces deux termes sont des synonymes de « sous-système » et proviennent du fait que typiquement, un sous-système planifie une ressource ou applique une limite aux groupes de contrôle dans la hiérarchie à laquelle il est attaché.

La définition d'un sous-système (ou contrôleur de ressources) est assez générale : il s'agit de quelquechose agissant sur un groupe de tâches, c'est-à-dire des processus.

1.2. RELATIONS ENTRE SOUS-SYSTÈMES, HIÉRARCHIES, GROUPES DE CONTRÔLE ET TÂCHES

Souvenez-vous que les processus système sont appelés des tâches dans la terminologie des groupes de contrôle.

Voici quelques simples règles régissant les relations entre sous-systèmes, hiérarchies des groupes de contrôle et tâches, ainsi que les conséquences explicatives de ces règles.

Règle n°1

Tout sous-système unique (comme **cpu**) peut être attaché à une seule hiérarchie au plus.

*En conséquence, le sous-système **cpu** ne pourra jamais être attaché à deux différentes hiérarchies.*

Règle n°2

Une hiérarchie unique peut avoir deux sous-systèmes ou plus qui lui sont attachés.

En conséquence, les sous-systèmes ***cpu*** *et* ***memory*** *(ou tout autre nombre de sous-systèmes) peuvent être attachés à une seule hiérarchie pourvu qu'aucun sous-système ne soit attaché à une autre hiérarchie.*

Règle n°3

Chaque fois qu'une nouvelle hiérarchie est créée sur les systèmes, toutes les tâches sur le système sont d'abord des membres du groupe de contrôle par défaut de cette hiérarchie, qui est aussi connu sous le nom de *groupe de contrôle racine* (de l'anglais, « root cgroup »). Pour toute hiérarchie unique que vous créez, chaque tâche sur le système peut être un membre d'*exactement un* groupe de contrôle dans cette hiérarchie. Une tâche unique peut se trouver dans de multiples groupes de contrôle, pourvu que chacun de ces groupes de contrôle se trouve dans une différente hiérarchie. Dès qu'une tâche devient membre d'un second groupe de contrôle dans la même hiérarchie, elle est supprimée du premier groupe de contrôle de cette hiérarchie. Une tâche ne peut jamais se trouver dans deux différents groupes de contrôle de le même hiérarchie.

Par conséquent, si les sous-systèmes ***cpu*** *et* ***memory*** *sont attachés à une hiérarchie nommée* ***cpu_and_mem*** *et si le sous-système* ***net_cls*** *est attaché à une hiérarchie nommée* ***net****, alors un processus* ***httpd*** *en cours d'exécution pourrait être un membre de n'importe quel groupe de contrôle se trouvant dans* ***cpu_and_mem*** *et de n'importe quel groupe de contrôle se trouvant dans* ***net****.*

Le groupe de contrôle dans ***cpu_and_mem*** *dont le processus* ***http*** *est un membre peut restreindre son temps processeur de moitié comparé à ce qui est alloué aux autres processus et limiter son utilisation de mémoire à un maximum de* ***1024*** *Mo. En outre, le groupe de contrôle dans* ***net*** *qui est un membre peut limiter son taux de transmission à* ***30*** *méga-octets par seconde.*

Lorsque la première hiérarchie est créée, chaque tâche sur le système est un membre d'au moins un groupe de contrôle : le groupe de contrôle racine (ou root). Ainsi, chaque tâche système se trouve au moins dans un groupe de contrôle.

Règle n°4

Tout processus (ou tâche) sur un système qui se copie lui-même donne naissance à un processus (tâche) enfant. La tâche enfant devient automatiquement membre de tous les groupes de contrôle dont le parent fait partie. La tâche enfant peut ensuite être déplacée sur différents groupes de contrôle selon les besoins, mais celle-ci hérite toujours des groupes de contrôle (de l'"environnement", pour reprendre la terminologie des processus) de sa tâche parente.

Par conséquent, prenez en considération la tâche ***httpd*** *qui est membre du groupe de contrôle nommé* ***half_cpu_1gb_max*** *dans la hiérarchie* ***cpu_and_mem*** *et du groupe de contrôle* ***trans_rate_30*** *dans la hiérarchie* ***net****. Lorsque ce processus* ***httpd*** *se copie lui-même, son processus enfant devient automatiquement membre des groupes de contrôle* ***half_cpu_1gb_max*** *et* ***trans_rate_30****. Celui-ci hérite des mêmes groupes de contrôle dont faisait partie sa tâche parente.*

À partir de ce moment, les tâches parent et enfant sont complètement indépendantes l'une de l'autre : modifier le groupe de contrôle auquel une tâche appartient n'affecte pas l'autre tâche. La modification des groupes de contrôle d'une tâche parent n'affectera aucune de ses tâches petits-enfants non plus. Pour résumer, toute tâche enfant commence par hériter des mêmes appartenances aux groupes de contrôle que ses tâches parentes, ces appartenances pourront être modifiées ou supprimées ultérieurement.

1.3. IMPLICATIONS POUR LA GESTION DES RESSOURCES

- Comme une tâche ne peut appartenir qu'à un seul groupe de contrôle dans une seule hiérarchie, il n'y a qu'une seule manière par laquelle une tâche peut être limitée ou affectée par un seul sous-système. Ceci est logique : il s'agit d'une fonctionnalité, et pas d'une limitation.

- Vous pouvez regrouper plusieurs sous-systèmes ensemble de manière à ce qu'ils affectent toutes les tâches dans une hiérarchie unique. Comme les groupes de contrôle dans cette hiérarchie possèdent de divers paramètres, ces tâches seront affectées de différemment.

- Il peut parfois être nécessaire de *refactoriser* une hiérarchie. Par exemple, en supprimant un sous-système d'une hiérarchie qui possède plusieurs sous-systèmes attachés, et en l'attachant à une nouvelle et différente hiérarchie.

- À l'inverse, si le besoin de diviser des sous-systèmes entre différentes hiérarchies est réduit, vous pouvez supprimer une hiérarchie et attacher ses sous-systèmes à une autre hiérarchie existante.

- Le design permet une utilisation facile d'un groupe de contrôle, comme la définition de quelques paramètres pour des tâches spécifiques dans une hiérarchie unique, comme lorsque seuls les sous-systèmes du processeur et de la mémoire sont attachés.

- Le design permet aussi de réaliser des configurations hautement spécifiques : chaque tâche (ou processus) sur un système peut être membre de chaque hiérarchie, qui elles-mêmes ne possèdent qu'un seul sous-système attaché. Une telle configuration pourrait ainsi offrir à un administrateur système un contrôle absolu sur tous les paramètres de chaque tâche.

---

[1] Le processus parent est capable d'altérer l'environnement avant de le passer à un processus enfant.

[2] Sachez que les sous-systèmes sont aussi appelés des *contrôleurs de ressources*, ou *contrôleurs*, dans les pages man libcgroup et les autres documents.

CHAPITRE 2. UTILISER LES GROUPES DE CONTRÔLE

La plus facile manière de travailler avec les groupes de contrôle est d'installer le paquetage libcgroup, qui contient un certain nombre d'utilitaires en ligne de commande qui sont liés aux groupes de contrôle et à leurs pages man associées. Il est possible de *monter* des hiérarchies et de définir des paramètres de groupes de contrôle (de manière non-persistante) à l'aide de commandes shell et d'utilitaires disponibles sur tout système. Cependant, utiliser les utilitaires fournis par libcgroup simplifie le processus et étend vos possibilités. Ainsi, ce guide se concentre principalement sur les commandes de libcgroup. Dans la plupart des cas, nous avons inclut les commandes shell équivalentes pour aider à décrire le mécanisme sous-jacent. Nous recommandons cependant d'utiliser les commandes libcgroup quand possible.

NOTE

Pour utiliser les groupes de contrôle, commencez par vous assurer que le paquetage libcgroup est installé sur votre système en exécutant la commande suivante en tant que super-utilisateur :

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

2.1. LE SERVICE CGCONFIG

Le service **cgconfig**, installé avec le paquetage libcgroup, fourni un moyen commode de créer des hiérarchies, d'attacher des sous-systèmes aux hiérarchies et de gérer les groupes de contrôle dans celles-ci. Nous recommandons d'utiliser **cgconfig** pour gérer les hiérarchies et les groupes de contrôle sur votre système.

Le service **cgconfig** n'est pas démarré par défaut sur Red Hat Enterprise Linux 6. Lorsque vous lancez le service avec **chkconfig**, il lit le fichier de configuration du groupe de contrôle — **/etc/cgconfig.conf**. Les groupes de contrôle sont ainsi recréés d'une session à l'autre et deviennent persistants. Selon le contenu du fichier de configuration, **cgconfig** peut créer des hiérarchies, monter les systèmes de fichiers nécessaires, créer des groupes de contrôle, et définir les paramètres de sous-système pour chaque groupe.

Le fichier par défaut **/etc/cgconfig.conf**, qui est installé avec le paquetage libcgroup crée et monte une hiérarchie individuelle pour chaque sous-système et attache les sous-systèmes à ces hiérarchies.

Si vous arrêtez le service **cgconfig** (avec la commande **service cgconfig stop**), celui-ci démontera toutes les hiérarchies qu'il avait monté.

2.1.1. Le fichier cgconfig.conf

Le fichier **/etc/cgconfig.conf** contient deux principaux types d'entrées — *mount* et *group*. Les entrées « mount » créent et montent les hiérarchies en tant que systèmes de fichiers virtuels et attachent les sous-systèmes à ces hiérarchies. Les entrées « mount » sont définies à l'aide de la syntaxe suivante :

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

Voir l'Exemple 2.1, « Création d'une entrée « mount » » pour un exemple d'utilisation.

Exemple 2.1. Création d'une entrée « mount »

L'exemple suivant crée une hiérarchies pour le sous-système **cpuset** :

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

équivaut aux commandes shell :

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

Les entrées « group » créent des groupes de contrôle et définissent les paramètres des sous-systèmes. Les entrées « group » sont définies à l'aide de la syntaxe suivante :

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

Remarquez que la section **permissions** est optionnelle. Pour définir les permissions pour une entrée « group », veuillez utiliser la syntaxe suivante :

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

Voir l'Exemple 2.2, « Création d'une entrée « group » » pour un exemple d'utilisation :

Exemple 2.2. Création d'une entrée « group »

L'exemple suivant crée un groupe de contrôle pour les démons sql, avec des permissions pour les utilisateurs faisant partie du groupe **sqladmin** afin d'ajouter des tâches au groupe de contrôle et pour l'utilisateur **root** afin de modifier les paramètres de sous-système :

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

```
        } admin {
            uid = root;
            gid = root;
        }
    } cpu {
        cpu.shares = 100;
    }
}
```

Lorsqu'elles sont combinées avec l'exemple de l'entrée « mount » dans l'Exemple 2.1, « Création d'une entrée « mount » », les commandes shelle équivalentes sont :

```
~]# 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
```

NOTE

Vous devez redémarrer le service **cgconfig** pour que les changements dans **/etc/cgconfig.conf** prennent effet :

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

Lorsque vous installez le paquetage libcgroup, un exemple de fichier de configuration est écrit sur **/etc/cgconfig.conf**. Les symboles dièse ('#'), se trouvant au début de chaque ligne en font des commentaires et les rendent invisibles au service **cgconfig**.

2.2. CRÉER UNE HIÉRARCHIE ET ATTACHER DES SOUS-SYSTÈMES

AVERTISSEMENT

Les instructions suivantes, qui couvrent la création d'une nouvelle hiérarchie et comment y attacher des sous-systèmes, suppose que les groupes de contrôle ne sont pas encore configurés sur votre système. Dans ce cas, ces instructions n'affecteront pas l'opération du système. La modification de paramètres réglables dans un groupe de contrôle avec des tâches pourrait cependant affecter ces tâches immédiatement. Ce guide vous prévient la première fois qu'il illustre la modification d'un paramètre de groupe de contrôle réglable qui pourrait affecter une ou plusieurs tâches.

Ces commandes échoueront sur un système où les groupes de contrôle sont déjà configurés (manuellement ou par le service **cgconfig**), à moins que vous ne démontiez d'abord les hiérarchies existantes, ce qui affectera l'opération du système. Ne pas expérimenter avec ces instructions sur des systèmes de production.

Pour créer un hiérarchie et y attacher des sous-systèmes, modifiez la section **mount** du fichier **/etc/cgconfig.conf** en tant que super-utilisateur. Les entrées de la section **mount** sont sous le format suivant :

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

Lorsque **cgconfig** démarrera la prochaine fois, il créera la hiérarchie et y attachera les sous-systèmes.

L'exemple suivant crée une hiérarchie nommée **cpu_and_mem** et y attache les sous-systèmes **cpu**, **cpuset**, **cpuacct** et **memory**.

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

Méthode alternative

Il est aussi possble d'utiliser des commandes shell et des utilitaires pour créer des hiérarchies et leur attacher des sous-système.

Créez un *point de montage* (*mount point*) pour la hiérarchie en tant que super-utilisateur. Incluez le nom du groupe de contrôle dans le point de montage :

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

Par exemple :

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

Puis utilisez la commande **mount** pour monter la hiérarchie et y attacher un ou plusieurs sous-système(s) simultanément. Par exemple :

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

Où *subsystems* est une liste séparée par virgules des sous-systèmes et *name* est le nom de la hiérarchie. De brèves descriptions de tous les sous-systèmes disponibles sont répertoriées dans les Sous-systèmes disponibles dans Red Hat Enterprise Linux et le Chapitre 3, *Sous-systèmes et paramètres réglables* fournit des références détaillées.

Exemple 2.3. Utilisation de la commande mount pour attacher des sous-systèmes

Dans cet exemple, un répertoire nommé **/cgroup/cpu_and_mem**, qui servira de point de montage pour la hiérarchie que nous créons, existe déjà. Nous attacherons les sous-systèmes **cpu**, **cpuset** et **memory** à une hiérarchie que nous nommons **cpu_and_mem**, puis nous montons (**mount**) la hiérarchie **cpu_and_mem** sur **/cgroup/cpu_and_mem** :

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

Il est possible de répertorier tous les sous-systèmes disponibles avec leurs points de montage actuels (c'est-à-dire l'emplacement où la hiérarchie à laquelle ils sont attachés est montée) avec la commande **lssubsys** [3] :

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

Cette sortie indique que :

- les sous-systèmes **cpu**, **cpuset** et **memory** sont attachés à une hiérarchie montée sur **/cgroup/cpu_and_mem**.
- les sous-systèmes **net_cls**, **ns**, **cpuacct**, **devices**, **freezer** et **blkio** ne sont toujours pas attaché à une hiérarchie, comme l'absence de points de montage le montre.

2.3. ATTACHER ET DÉTACHER DES SOUS-SYSTÈMES D'UNE HIÉRARCHIE EXISTANTE

Pour ajouter un sous-système à une hiérarchie existante, le détacher d'une hiérarchie existante, ou pour le déplacer sur une autre hiérarchie, modifiez la section **mount** du fichier **/etc/cgconfig.conf** en tant que super-utilisateur en utilisant la même syntaxe que celle décrite dans la Section 2.2, « Créer une hiérarchie et attacher des sous-systèmes ». La prochaine fois que **cgconfig** démarrera, il réorganisera les sous-systèmes en fonction des hiérarchies spécifiées.

Méthode alternative

Pour ajouter un sous-système non attaché à une hiérarchie existante, montez la hiérarchie à nouveau. Veuillez inclure le sous-système supplémentaire dans la commande **mount** ainsi que l'option **remount**.

Exemple 2.4. Monter une hiérarchie à nouveau pour ajouter un sous-système

La commande **lssubsys** affiche les sous-systèmes **cpu**, **cpuset** et **memory**, qui sont attachés à la hiérarchie **cpu_and_mem** :

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

Montons la hiérarchie **cpu_and_mem** à nouveau, en utilisant l'option **remount** et en incluant **cpuacct** dans la liste des sous-systèmes :

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

La commande **lssubsys** affiche maintenant que **cpuacct** est attaché à la hiérarchie **cpu_and_mem** :

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

De manière analogue, il est possible de détacher un sous-système d'une hiérarchie existante en mont la hiérarchie à nouveau et en omettant le nom du sous-système dans les options -o. Par exemple, pour détacher le sous-système **cpuacct**, montez-le à nouveau puis omettez-le :

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

2.4. DÉMONTER UNE HIÉRARCHIE

Vous pouvez *démonter* (*unmount*) une hiérarchie de groupes de contrôle avec la commande **umount** :

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

Par exemple :

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

Si la hiérarchie est actuellement vide (c'est-à-dire si elle ne contient que le groupe de contrôle racine), alors celle-ci est désactivée lorsqu'elle est démontée. Si la hiérarchie contient tout autre groupe de contrôle, alors elle restera active dans le noyau même si elle n'est plus montée.

Pour supprimer une hiérarchie, assurez-vous que tous les groupes de contrôle enfants sont supprimés avant de la démonter, ou utilisez la commande **cgclear**, qui désactive une hiérarchie même si celle-ci n'est pas vide — reportez-vous à la Section 2.11, « Décharger les groupes de contrôle ».

2.5. CRÉATION DE GROUPES DE CONTRÔLE

Utilisez la commande **cgcreate** pour créer des groupes de contrôle. La syntaxe pour **cgcreate** est : **cgcreate -t *uid*:*gid* -a *uid*:*gid* -g *subsystems*:*path*** , où :

- **-t** (optionnel) — spécifie un utilisateur (par un ID utilisateur, ou uid) et un groupe (par ID de groupe, ou GID) pour posséder le pseudofichier **tâches** (**tasks**) pour ce groupe de contrôle. Cet utilisateur peut ajouter des tâches à ce groupe de contrôle.

NOTE

Remarquez que l'unique manière de supprimer une tâche d'un groupe de contrôle est de la déplacer sur un autre groupe de contrôle. Pour déplacer une tâche, l'utilisateur doit posséder la permission d'écriture sur le groupe de contrôle *destinataire* ; la permission d'écriture sur le groupe de contrôle source n'est pas importante.

- **-a** (optionnel) — spécifie un utilisateur (par un ID utilisateur, ou uid) et un groupe (par ID de groupe, ou GID) pour posséder tous les pseudofichiers autres que **tâches** (**tasks**) pour ce groupe de contrôle. Cet utilisateur peut modifier les accès des tâches de ce groupe de contrôle aux ressources système.

- **-g** — spécifie la hiérarchie dans laquelle le groupe de contrôle devrait être créé, en tant que liste séparée par des virgules des *sous-systèmes* associés à ces hiérarchies. Si les sous-systèmes de cette liste se trouvent dans différentes hiérarchies, le groupe est créé dans chacune d'entre elles. La liste des hiérarchies est suivie par deux-points et le *chemin d'accès* vers le groupe enfant relatif à la hiérarchie. Ne pas inclure le point de montage de la hiérarchie dans le chemin d'accès.

  Par exemple, le groupe de contrôle se trouvant dans le répertoire **/cgroup/cpu_and_mem/lab1/** est simplement appelé **lab1** — son chemin d'accès est déjà déterminé de manière unique car il y a, au plus, une seule hiérarchie pour un sous-système donné. Remarquez aussi que le groupe est contrôlé par tous les sous-systèmes qui existent dans les hiérarchies dans lesquelles le groupe de contrôle a été créé, même si ces sous-systèmes n'ont pas été spécifiés dans la commande **cgcreate** — reportez-vous à la Exemple 2.5, « Utilisation de cgcreate ».

Comme tous les groupes de contrôle dans la même hiérarchie possèdent les mêmes contrôleurs, le groupe enfant possède les mêmes contrôleurs que son parent.

Exemple 2.5. Utilisation de cgcreate

Prenez en considération un système où les sous-systèmes **cpu** et **memory** sont montés ensemble dans la hiérarchie **cpu_and_mem** et où le contrôleur **net_cls** est monté dans une hiérarchie séparée nommée **net**. Nous exécutons mainenant :

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

La commande **cgcreate** crée deux groupes nommés **test-subgroup**, un dans la hiérarchie **cpu_and_mem** et un dans la hiérarchie **net**. Le groupe **test-subgroup** dans la hiérarchie **cpu_and_mem** est contrôlé par le sous-système **memory**, même si nous ne l'avons pas spécifié dans la commande **cgcreate**.

Méthode alternative

Pour créer un enfant du groupe de contrôle, utilisez la commande **mkdir** :

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

Par exemple :

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

2.6. SUPPRIMER DES GROUPES DE CONTRÔLE

Il est possible de supprimer les groupes de contrôle avec **cgdelete**, qui possède une syntaxe similaire à celle de **cgcreate**. Veuillez exécuter : **cgdelete *subsystems*:*path***, où :

- *subsystems* est une liste des sous-systèmes séparée par des virgules.

- *path* est le chemin d'accès vers le groupe de contrôle relatif à la racine (root) de la hiérarchie.

Par exemple :

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

cgdelete peut aussi supprimer récursivement tous les sous-groupes avec l'option **-r**.

Lorsque vous supprimez un groupe de contrôle, toutes ses tâches se déplacent sur son groupe parent.

2.7. DÉFINIR LES PARAMÈTRES

Il est possible de définir les paramètres des sous-systèmes en exécutant la commande **cgset** depuis un compte utilisateur avec la permission de modifier le groupe de contrôle approprié. Par exemple, si **/cgroup/cpuset/group1** existe, spécifiez les CPUs auxquels ce groupe aura accès avec la commande suivante :

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

La syntaxe pour **cgset** est : **cgset -r *parameter=value path_to_cgroup*** , où :

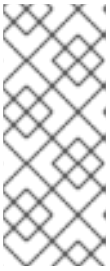
- *parameter* est le paramètre à définir qui correspond au fichier dans le répertoire du groupe de contrôle donné
- *value* est la valeur du paramètre
- *path_to_cgroup* est le chemin d'accès au groupe de contrôle *relatif à la racine de la hiérarchie*. Par exemple, pour définir le paramètre du groupe racine (si **/cgroup/cpuacct/** existe), exécutez :

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

  Alternativement, comme **.** est relatif au groupe racine (le groupe racine lui-même), vous pourriez aussi exécuter :

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

  Remarquez cependant que **/** est la syntaxe préférée.

  NOTE

  Seul un petit nombre de paramètres peuvent être définis pour le groupe racine (comme le paramètre **cpuacct.usage** montré dans les exemples ci-dessus). Ceci est dû au fait qu'un groupe racine est propriétaire de toutes les ressources existantes. Ainsi, il ne servirait à rien de limiter tous les processus existants en définissant certains paramètres, le paramètre **cpuset.cpu** par exemple.

  Pour définir le paramètre de **group1**, qui est un sous-groupe du groupe racine, exécutez :

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

Une barre oblique finale après le nom du groupe est optionnelle (par exemple, **cpuacct.usage=0 group1/**).

Les valeurs que vous pouvez définir avec **cgset** peuvent dépendre des valeurs définies plus haut dans la hiérarchie. Par exemple, si **group1** est limité à l'unique utilisation de CPU 0 sur un système, vous ne pourrez pas définir **group1/subgroup1** pour qu'il utilise CPUs 0 et 1, ou pour qu'il utilise uniquement CPU 1.

Vous pouvez aussi utiliser **cgset** pour copier les paramètres d'un groupe de contrôle à un autre groupe de contrôle existant. Par exemple :

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

La syntaxe pour copier des paramètres avec **cgset** est : **cgset --copy-from *path_to_source_cgroup path_to_target_cgroup***, où :

- *path_to_source_cgroup* est le chemin d'accès au groupe de contrôle dont les paramètres sont à copier de manière relative au groupe racine de la hiérarchie
- *path_to_target_cgroup* est le chemin d'accès au groupe de contrôle destinataire relatif au groupe racine de la hiérarchie

Assurez-vous que tous les paramètres obligatoires pour les divers sous-systèmes soient bien installés avant de copier des paramètres d'un groupe à un autre, sinon la commande échouera. Pour obtenir plus d'informations sur les paramètres obligatoires, reportez-vous à Important — Paramètres obligatoires.

Méthode alternative

Pour définir des paramètres dans un groupe de contrôle, insérez les valeurs dans le pseudofichier du sous-système correspondant à l'aide de la commande **echo**. Par exemple, cette commande insère la valeur **0-1** dans le pseudofichier **cpuset.cpus** du groupe de contrôle **group1** :

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

Avec cette valeur en place, les tâches de ce groupe de contrôle sont restreintes aux CPUs 0 et 1 du système.

2.8. DÉPLACER UN PROCESSUS SUR UN GROUPE DE CONTRÔLE

Vous pouvez aussi déplacer un processus en exécutant la commande **cgclassify** :

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

La syntaxe pour **cgclassify** est : **cgclassify -g *subsystems*:*path_to_cgroup pidlist***, où :

- *subsystems* est une liste des sous-systèmes séparée par des virgules, ou ***** pour lancer le processus dans les hiérarchies associées à tous les sous-systèmes disponibles. Remarquez que si des groupes de contrôle du même nom existent dans de multiples hiérarchies, l'option **-g** déplace les processus dans chacun de ces groupes. Assurez-vous que le groupe de contrôle existe dans chacune des hiérarchies dont vous spécifiez les sous-systèmes ici.
- *path_to_cgroup* est le chemin vers le groupe de contrôle dans ses hiérarchies
- *pidlist* est une liste d'*identifiants de processus* (ou PID) séparée par des espace

Vous pouvez aussi ajouter l'option **-- sticky** avant le *pid* afin de conserver tout processus enfant dans le même groupe de contrôle. Si vous ne spécifiez pas cette option et que le démon **cgred** est en cours d'exécution, les processus enfants seront alloués aux groupes de contrôle selon les paramètres se trouvant dans **/etc/cgrules.conf**. Le processus, quant à lui, restera dans le groupe de contrôle à partir duquel vous l'avez démarré.

À l'aide de **cgclassify**, vous pouvez déplacer plusieurs processus simultanément. Par exemple, cette commande déplace les processus avec les PIDs **1701** et **1138** dans le groupe de contrôle **group1/** :

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

Remarquez que les PIDs à déplacer sont séparés par des espaces et que les groupes spécifiés doivent être dans différentes hiérarchies.

Méthode alternative

Pour déplacer un processus vers un groupe de contrôle, écrivez son PID sur le fichier des **tâches** (**tasks**) du groupe de contrôle. Par exemple, pour déplacer un processus avec le PID **1701** dans un groupe de contrôle sur **/cgroup/lab1/group1/** :

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

2.8.1. Le démon cgred

Cgred est un démon qui déplace les tâches dans des groupes de contrôle en fonction des paramètres définis dans le fichier **/etc/cgrules.conf**. Les entrées du fichier **/etc/cgrules.conf** peuvent prendre l'une des formes suivantes :

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

Par exemple :

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

Cette entrée spécifie que tout processus appartenant à l'utilisateur nommé **maria** peuvent accéder au sous-système des périphériques selon les paramètres spécifiés dans le groupe de contrôle **/usergroup/staff**. Pour associer des commandes particulières à des groupes de contrôle particuliers, ajoutez le paramètre *command* comme suit :

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

L'entrée spécifie maintenant que lorsque l'utilisateur nommé **maria** utilise la commande **ftp**, le processus est automatiquement déplacé sur le groupe de contrôle **/usergroup/staff/ftp** dans la hiérarchie qui contient le sous-système **devices**. Remarquez cependant que le démon déplace le processus sur le groupe de contrôle uniquement après que la condition appropriée aura été remplie. Ainsi, le processus **ftp** peut être exécuté pendant un court moment dans le mauvais groupe. En outre, si le processus génère des enfants alors qu'il se trouve dans le mauvais groupe, ces enfants pourraient ne pas être déplacés.

Les entrées dans le fichier **/etc/cgrules.conf** peuvent inclure les notations supplémentaire suivantes :

- @ — lorsque préfixé à *user*, ceci indique un groupe au lieu d'un utilisateur unique. Par exemple, **@admins** signale tous les utilisateurs faisant partie du groupe **admins**.
- ***** — représente "tout". Par exemple, ***** dans le champ **sous-système** représente tous les sous-systèmes.
- % — représente un élément qui est le même que l'élément présent dans la ligne du dessus. Par exemple :

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

2.9. LANCER UN PROCESSUS DANS UN GROUPE DE CONTRÔLE

IMPORTANT

Certains sous-systèmes ont des paramètres obligatoires à définir avant de déplacer une tâche dans un groupe de contrôle qui utilise l'un de ces sous-systsèmes. Par exemple, avant de pouvoir déplacer une tâche dans un groupe de contrôle qui utilise le sous-système **cpuset**, les paramètres **cpuset.cpus** et **cpuset.mems** doivent être définis pour ce groupe de contrôle.

Les exemples dans cette section illustrent la syntaxe correcte pour la commande, mais ne fonctionnent que sur les systèmes sur lesquels les paramètres obligatoires ont été définis pour tout contrôleur utilisé dans les exemples. Si vous n'avez pas déjà configuré les contrôleurs appropriés, vous ne pourrez pas copier les commandes d'exemple directement depuis cette section et vous attendre à ce qu'elles fonctionnent sur votre système.

Reportez-vous à la Section 3.10, « Ressources supplémentaires » pour obtenir une description des paramètres obligatoires aux sous-systèmes donnés.

Vous pouvez aussi lancer des processus dans un groupe de contrôle en exécutant la commande **cgexec**. Par exemple, cette commande lance le navigateur web **lynx** dans le groupe de contrôle **group1**, sujet à toutes les limitations imposées par le sous-système **cpu** :

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

La syntaxe pour **cgexec** est : **cgexec -g *subsystems:path_to_cgroup command arguments*** , où :

- *subsystems* est une liste des sous-systèmes séparée par des virgules, ou ***** sert à lancer le processus dans les hiérarchies associées à tous les sous-systèmes disponibles. Remarquez que comme pour **cgset**, qui est décrit dans la Section 2.7, « Définir les paramètres », si des groupes de contrôle de même nom existent dans de multiples hiérarchies, l'option **-g** crée des processus dans chacun de ces groupes. Assurez-vous que le groupe de contrôle existe bien dans cahcune des hiérarchies dont vous spécifiez les sous-systèmes.
- *path_to_cgroup* est le chemin d'accès vers le groupe de contrôle relatif à la hiérarchie.
- *command* est la commande à exécuter
- *arguments* représente tout argument de la commande

Vous pouvez aussi ajouter l'option **-- sticky** avant *command* afin de conserver tout processus enfant dans le même groupe de contrôle. Si vous ne saisissez pas cette option et que le démon **cgred** est en cours d'exécution, les processus enfants seront alloués aux groupes de contrôle en fonction des paramètres trouvés sur **/etc/cgrules.conf**. Le processus, toutefois, restera dans le groupe de contrôle à partir duquel il a été démarré.

Méthode alternative

Lorsque vous lancez un nouveau processus, il hérite du groupe de son processus parent. Ainsi, déplacer votre processus shell sur un groupe de contrôle puis lancer le processus à partir de ce shell constitue une méthode alternative pour démarrer un processus dans un groupe de contrôle particulier (reportez-vous à la Section 2.8, « Déplacer un processus sur un groupe de contrôle »). Par exemple :

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

Remarquez qu'après la fermeture de **lynx**, le shell existant se trouve toujours dans le groupe de contrôle **group1**. Ainsi, la méthode suivante est encore meilleure :

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

2.9.1. Lancer un service dans un groupe de contrôle

Il est possible de démarrer certains services dans un groupe de contrôle. Les services pouvant être démarrés dans des groupes de contrôle doivent :

- utiliser un fichier **/etc/sysconfig/*servicename***
- utiliser la fonction **daemon()** depuis **/etc/init.d/functions** pour démarrer le service

Pour lancer un service dans un groupe de contrôle, modifiez son fichier dans le répertoire **/etc/sysconfig** de manière à inclure une entrée sous la forme **CGROUP_DAEMON="*subsystem*:*control_group*"**, où *subsystem* est un sous-système associé à une hiérarchie particulière, et où *control_group* est un groupe de contrôle dans cette hiérarchie. Par exemple :

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

2.10. OBTENIR DES INFORMATIONS SUR LES GROUPES DE CONTRÔLE

2.10.1. Trouver un processus

Pour trouver le groupe de contrôle auquel un processus appartient, exécutez :

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

Ou bien, si vous connaissez le PID du processus, exécutez :

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

2.10.2. Trouver un sous-système

Pour trouver les sous-systèmes qui sont disponibles dans le noyau et de quelle manière ils sont montés ensembles sur des hiérarchies, exécutez :

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

Sinon, pour trouver les points de montage de sous-systèmes particuliers, exécutez :

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

ou *subsystems* est une liste des sous-systèmes qui vous intéresserait. Remarquez que la commande **lssubsys -m** retourne uniquement le point de montage le plus haut de chaque hiérarchie.

2.10.3. Trouver des hiérarchies

Il est recommandé de monter les hiérarchies sous **/cgroup**. En supposant que ce soit le cas sur votre système, répertoriez ou naviguez sur le contenu de ce répertoire pour obtenir une liste des hiérarchies. Si **tree** est installé sur votre système, exécutez-le pour obtenir une vue d'ensemble de toutes les hiérarchies et de tous les groupes de contrôle se trouvant dedans :

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

2.10.4. Trouver des groupes de contrôle

Pour répertorier les groupes de contrôle sur un système, exécutez :

```
~]$ lscgroup
```

Il est possible de restreindre la sortie à une hiérarchie spécifique en précisant un contrôleur et un chemin sous le format ***controller*:*path***. Par exemple :

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

répertorie uniquement les sous-groupes du groupe de contrôle **adminusers** dans la hiérarchie à laquelle le sous-système **cpuset** est attaché.

2.10.5. Afficher les paramètres des groupes de contrôle

Pour afficher les paramètres de groupes de contrôle spécifiques, exécutez :

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

où *parameter* est un pseudofichier contenant les valeurs d'un sous-système et *list_of_cgroups* est une liste des groupes de contrôle séparée par des espaces. Par exemple :

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

affiche les valeurs de **cpuset.cpus** et **memory.limit_in_bytes** des groupes de contrôle **lab1** et **lab2**.

Si vous ne connaissez pas les noms des paramètres, utilisez une commande comme :

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

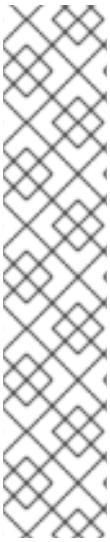
2.11. DÉCHARGER LES GROUPES DE CONTRÔLE

AVERTISSEMENT

La commande **cgclear** supprime tous les groupes de contrôle dans toutes les hiérarchies. Si vous n'avez pas stocké ces hiérarchies dans un fichier de configuration, vous ne serez pas en mesure de les reconstruires.

Pour supprimer un système de fichiers de groupe de contrôle entier, utilisez la commande **cgclear**.

Toutes les tâches dans le groupe de contrôle sont réaffectées dans le noeud racine (root) des hiérarchies, tous les groupes de contrôle sont supprimés, et le système de fichiers est démonté du système, détruisant ainsi toutes les hiérarchies précédemment montées. Finalement, le répertoire où le système de fichiers du groupe de contrôle était monté est supprimé.



NOTE

L'utilisation de la commande **mount** pour créer des groupes de contrôle (contrairement à leur création à l'aide du service **cgconfig**) résulte en la création d'une entrée dans le fichier **/etc/mtab** (le tebleau des systèmes de fichiers montés). Ce changement est aussi reflété dans le fichier **/proc/mounts**. Cependant, le déchargement des groupes de contrôle avec la commande **cgclear**, ainsi qu'avec d'autres commandes **cgconfig**, utilise une interface de noyau directe qui ne reflète pas ses changements dans le fichier **/etc/mtab** et qui écrit les nouvelles informations uniquement dans le fichier **/proc/mounts**. Ainsi, après avoir déchargé les groupes de contrôle avec la commande **cgclear**, les groupes de contrôle non montés pourraient toujours être visibles sur le fichier **/etc/mtab**, et par conséquent, ils seront affichés lorsque la commande **mount** est exécutée. Il est préférable de se reporter au fichier **/proc/mounts** pout obtenir une liste précise des tous les groupes de contrôle montés.

2.12. RESSOURCES SUPPLÉMENTAIRES

La documentation finale pour les commande des groupes de contrôle se trouve dans les pages man fournies avec le paquetage libcgroup. Les numéros des sections sont spécifiés dans la liste des pages man ci-dessous.

Les pages man libcgroup

- **man 1 cgclassify** — la commande **cgclassify** est utilisée pour déplacer des tâches en cours d'exécution sur un ou plusieurs groupe(s) de contrôle.

  man 1 cgclear — la commande **cgclear** est utilisée pour supprimer tous les groupes de contrôle dans une hiérarchie.

man 5 cgconfig.conf — les groupes de contrôle sont définis dans le fichier **cgconfig.conf**.

man 8 cgconfigparser — la commande **cgconfigparser** analyse le fichier **cgconfig.conf** et monte les hiérarchies.

man 1 cgcreate — la commande **cgcreate** crée de nouveaux groupes de contrôle dans les hiérarchies.

man 1 cgdelete — la commande **cgdelete** supprime les groupes de contrôles spécifiés.

man 1 cgexec — la commande **cgexec** exécute des tâches dans les groupes de contrôle spécifiés.

man 1 cgget — la commande **cgget** affiche les paramètres des groupes de contrôle.

man 5 cgred.conf — **cgred.conf** est le fichier de configuration du service **cgred**.

man 5 cgrules.conf — le fichier **cgrules.conf** contient les règles utilisées pour déterminer les tâches qui appartiennent à certains groupes de contrôle.

man 8 cgrulesengd — le service **cgrulesengd** distribue les tâches aux groupes de contrôle.

man 1 cgset — la commande **cgset** définit les paramètres d'un groupe de contrôle.

man 1 lscgroup — la commande **lscgroup** répertorie les groupes de contrôle dans une hiérarchie.

man 1 lssubsys — la commande **lssubsys** répertorie les hiérarchies contenant des sous-systèmes spécifiés.

---

[3] La commande **lssubsys** est l'un des utilitaires fournis par le paquetage libcgroup. Veuillez installer libcgroup pour l'utiliser : reportez-vous au Chapitre 2, *Utiliser les groupes de contrôle* si vous ne parvenez pas à exécuter **lssubsys**.

CHAPITRE 3. SOUS-SYSTÈMES ET PARAMÈTRES RÉGLABLES

Les *sous-systèmes* sont des modules du noyau prenant en charge les groupes de contrôle (cgroups). Il s'agit typiquement de contrôleurs de ressources qui allouent différents niveaux de ressources système à différents groupes de contrôle. Cependant, les sous-systèmes pourraient être programmés pour toute autre interaction avec le noyau si le besoin de traiter les divers groupes de processus différemment existe. L'*interface de programmation* (API) servant à développer de nouveaux sous-systèmes est documentée dans **`cgroups.txt`** dans la documentation du noyau qui est installée sur **`/usr/share/doc/kernel-doc-`*`kernel-version`*`/Documentation/cgroups/`** (fourni par le paquetage kernel-doc). La version la plus récente de la documentation des groupes de contrôle est disponible en ligne à l'adresse suivante : http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt. Remarquez cependant que les fonctionnalités décrites dans les documents les plus récents pourraient ne pas correspondre à celles qui sont disponibles dans le noyau installé sur votre système.

Les *objets d'état* qui contiennent les paramètres de sous-système pour un groupe de contrôle (cgroup) sont représentés en tant que *pseudofichiers* dans le système de fichiers virtuel du groupe de contrôle. Ces pseudofichiers peuvent être manipulés à l'aide de commandes shell ou des appels système équivalents. Par exemple, **`cpuset.cpus`** est un pseudofichier qui spécifie à quels CPUs un groupe de contrôle à l'autorisation d'accès. Si **`/cgroup/cpuset/webserver`** est un groupe de contrôle pour le serveur web exécuté sur un système et que nous exécutons la commande suivante :

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

La valeur **`0,2`** est écrite sur le pseudofichier **`cpuset.cpus`** et limite ainsi toutes les tâches dont les PID sont répertoriés sur **`/cgroup/cpuset/webserver/tasks`** afin d'uniquement utiliser CPU 0 et CPU 2 sur le système.

3.1. BLKIO

Le sous-système Block I/O (**`blkio`**) surveille et contrôle l'accès des tâches de groupes de contrôle aux E/S sur des périphériques blocs. L'écriture de valeurs sur certains de ces pseudo-fichiers limite l'accès ou la bande passante, la lecture des valeurs de certains de ces pseudos-fichiers fournit des informations sur les opérations d'E/S.

blkio.weight

spécifie la proportion relative (*poids*) de l'accès de Block I/O disponible par défaut à un groupe de contrôle dans une gamme de **`100`** à **`1000`**. Cette valeur est remplacée par le paramètre **`blkio.weight_device`** pour des périphériques spécifiques. Par exemple, pour assigner un poids par défaut de **`500`** à un groupe de contrôle pour accéder aux périphériques blocs, exécutez :

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

blkio.weight_device

spécifie la proportion relative (*poids*) de l'accès des E/S à un groupe de contrôle sur des périphériques spécifiques dans la gamme de **`100`** à **`1000`**. La valeur de ce paramètre remplace la valeur du paramètre **`blkio.weight`** pour des périphériques spécifiés. Les valeurs prennent le format *majeur*:*mineur poids*, où *majeur* et *mineur* sont des types de périphériques et des numéros de noeuds spécifiés dans les *Périphériques alloués Linux*, aussi connus sous le nom de *Liste des périphériques Linux* et disponible sur http://www.kernel.org/doc/Documentation/devices.txt. Par exemple, pour assigner un poids de **`500`** à un groupe de contrôle pour accéder à **`/dev/sda`**, exécutez :

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

Dans la notation des *Périphériques alloués Linux*, **8:0** représente **/dev/sda**.

blkio.time

rapporte le moment auquel un groupe de contrôle possédait un accès d'E/S à des périphériques spécifiques. Les entrées possèdent trois champs : *majeur*, *mineur* et *durée*. *Majeur* et *mineur* sont des types de périphériques et des numéros de noeuds spécifiés dans les *Périphériques alloués Linux* et *durée* est la durée en millisecondes (ms).

blkio.sectors

rapporte le nombre de secteurs transférés depuis ou vers des périphériques spécifiques d'un groupe de contrôle. Les entrées possèdent trois champs : *majeur*, *mineur* et *secteurs*. *Majeur* et *mineur* sont des types de périphériques et des numéros de noeuds spécifiés dans les *Périphériques alloués Linux* et *secteurs* est le nombre de secteurs du disque.

blkio.io_service_bytes

rapporte le nombre d'octets transférés depuis ou vers des périphériques spécifiques par un groupe de contrôle. Les entrées possèdent quatre champs : *majeur*, *mineur*, *opération* et *octets*. *Majeur* et *mineur* sont les types de périphériques et des numéros de noeuds spécifiés dans les *Périphériques alloués Linux*, *opération* représente le type d'opération (**lecture**, **écriture**, **sync** ou **async**) et *octets* est le nombre d'octets transférés.

blkio.io_serviced

rapporte le nombre d'opérations d'E/S effectuées sur des périphériques spécifiques par un groupe de contrôle. Les entrées possèdent quatre champs : *majeur*, *mineur*, *opération* et *octets*. *Majeur* et *mineur* sont les types de périphériques et numéros de noeuds spécifiés dans les *Périphériques alloués Linux*, *opération* représente le type d'opération (**lecture**, **écriture**, **sync** ou **async**) et *nombre* représente le nombre d'opérations.

blkio.io_service_time

rapporte la totalité du temps pris entre l'envoi de la demande et l'achèvement de celle-ci pour des opérations d'E/S sur des périphériques spécifiques par un groupe de contrôle. Les entrées possèdent quatre champs : *majeur*, *mineur*, *opération* et *octets*. *Majeur* et *mineur* sont les types de périphériques et numéros de noeuds spécifiés dans les *Périphériques alloués Linux*, *opération* représente le type d'opération (**lecture**, **écriture**, **sync**, ou **async**) et *durée* représente la durée en nanosecondes (ns). La durée rapportée utilise des nanosecondes plutôt qu'une unité plus importante afin que ce rapport puisse être utile même lors de l'utilisation de périphériques SSD.

blkio.io_wait_time

rapporte le temps total attendu pour un service dans les files d'attente des opérations d'E/S sur des périphériques spécifiques d'un groupe de contrôle. Lorsque vous interprétez ce rapport, remarquez que :

- le temps rapporté peut être plus important que la totalité du temps écoulé car le temps rapporté est le total cumulé des toutes les opérations d'E/S pour le groupe de contrôle plutôt que le temps que le groupe de contrôle a lui-même attendu pour les opérations d'E/S. Pour découvrir combien de temps le groupe en tant que tel a dû attendre, utilisez **blkio.group_wait_time**.

- si **queue_depth** > 1 (sur le périphérique), alors la durée rapportée inclut uniquement le temps attendu jusqu'à ce que la requête soit envoyée au périphérique, et pas le temps

attendu pour le service pendant que le périphériques effectuait de nouvelles commandes de requêtes.

Les entrées possèdent quatre champs : *majeur*, *mineur*, *opération* et *octets*. *Majeur* et *mineur* sont les types de périphériques et numéros de noeuds spécifiés dans les *Périphériques alloués Linux*, *opération* (**lecture**, **écriture**, **sync** ou **async**) et *durée* est la durée en nanosecondes (ns). La durée rapportée utilise des nanosecondes plutôt qu'une unité plus importante afin que ce rapport puisse être utile même pour des périphériques SSD.

blkio.io_merged

rapporte le nombre de requêtes BIOS fusionnées en requêtes pour des opérations d'E/S d'un groupe de contrôle. Les entrées possèdent deux champs : *nombre* et *opération*. *Nombre* est le nombre de requêtes et *opération* représente le type d'opération (**lecture**, **écriture**, **sync**, ou **async**).

blkio.io_queued

rapporte le nombre de requêtes en file d'attente pour des opérations d'E/S d'un groupe de contrôle. Les entrées possèdent deux champs : *nombre* et *opération*. *Nombre* est le nombre de requêtes et *opération* représente le type d'opération (**lecture**, **écriture**, **sync** ou **async**).

blkio.avg_queue_size

rapporte la taille moyenne des files d'attente pour des opérations d'E/S d'un groupe de contrôle sur la durée totale de l'existance du groupe. La taille de la file d'attente est échantillonnée chaque fois qu'une file d'attente de ce groupe de contrôle obtient une tranche de temps. Remarquez que ce rapport est uniquement disponible si **CONFIG_DEBUG_BLK_CGROUP=y** est installé sur le système.

blkio.group_wait_time

rapporte la durée totale (en nanosecondes — ns) qu'un groupe de contrôle a attendu une tranche de temps pour l'une de ses files d'attente. Le rapport est mis à jour à chaque fois qu'une file d'attente de ce groupe de contrôle obtient une tranche de temps. Ainsi, si vous lisez ce pseudo-fichier alors que le groupe de contrôle attend sa tranche de temps, le rapport ne contiendra pas le temps attendu pour l'opération se trouvant actuellement en file d'attente. Remarquez que ce rapport est uniquement disponible si **CONFIG_DEBUG_BLK_CGROUP=y** est installé sur le système.

blkio.empty_time

rapporte la durée totale (en nanosecondes — ns) qu'un groupe de contrôle a attendu sans requête en attente. Le rapport est mis à jour à chaque fois qu'une file d'attente de ce groupe de contrôle possède une requête en attente. Ainsi, si vous lisez ce pseudo-fichier alors que le groupe de contrôle ne possède pas de requêtes en attente, le rapport ne contiendra pas le temps attendu dans l'état actuel. Remarquez que ce rapport est uniquement disponible si **CONFIG_DEBUG_BLK_CGROUP=y** est installé sur le système.

blkio.idle_time

rapporte le temps total (en nanosecondes — ns) que le planificateur a passé à attendre un groupe de contrôle dans l'attente d'une meilleure requête que les requêtes se trouvant déjà dans des files d'attente ou provenant d'autres groupes. Le rapport est mis à jour à chaque fois qu'un groupe redevient actif. Ainsi, si vous lisez ce pseudo-fichier alors que le groupe de contrôle est inactif (idling), le rapport ne contiendra pas le temps passé à attendre dans l'état inactif (idling) actuel. Remarquez que ce rapport est uniquement disponible si **CONFIG_DEBUG_BLK_CGROUP=y** est installé sur le système.

blkio.dequeue

rapporte le nombre de fois que des requêtes d'opérations d'E/S d'un groupe de contrôle ont été retirées des files d'attente par des périphériques spécifiques. Les entrées possèdent trois champs : *majeur*, *mineur* et *nombre*. *Majeur* et *mineur* sont les types de périphérique et numéros de noeud spécifiés dans les *Périphériques alloués Linux* et *nombre* est le nombre de requêtes que le groupe a retiré de la file d'attente. Remarquez que ce rapport est uniquement disponible si **CONFIG_DEBUG_BLK_CGROUP=y** est installé sur le système.

blkio.reset_stats

réinitialise les statistiques enregistrées dans les autres pseudo-fichiers. Saisissez un entier sur ce fichier pour réinitialiser les statistiques de ce groupe de contrôle.

3.2. CPU

Le sous-système **cpu** planifie l'accès du CPU aux groupes de contrôle. L'accès aux ressources CPU peut être planifié en fonction des paramètres suivants, chacun dans un fichier *pseudofile* (un pseudo-fichier) à l'intérieur du système de fichiers virtuels du groupe de contrôle :

cpu.shares

contient une valeur entière spécifiant une part relative du temps du CPU disponible pour les tâches dans un groupe de contrôle (un cgroup). Par exemple, des tâches dans deux groupes de contrôle possédant **cpu.shares** paramétré sur **1** recevront le même temps de CPU, mais des tâches dans un groupe de contrôle possédant **cpu.shares** paramétré sur **2** recevront deux fois plus de temps de CPU que les tâches où **cpu.shares** est paramétré sur **1**.

cpu.rt_runtime_us

spécifie une période de temps en microsecondes (µs, représenté ici sous la forme "us") pour la plus longue période continue pendant laquelle les tâches d'un groupe de contrôle ont accès aux ressources du CPU. Établir cette limite permet d'empêcher les tâches d'un groupe de contrôle de monopoliser le temps du CPU. Si les tâches dans un groupe de contrôle doivent pouvoir accéder aux ressources du CPU pendant 4 secondes sur chaque période de 5 secondes, ajustez **cpu.rt_runtime_us** sur **4000000** et **cpu.rt_period_us** sur **5000000**.

cpu.rt_period_us

spécifie une période de temps en microsecondes (µs, représenté ici sous la forme "us") pour définir à quelle fréquence réallouer l'accès d'un groupe de contrôle aux ressources du CPU. Si les tâches d'un groupe de contrôle doivent pouvoir accéder aux ressources du CPU pendant 4 secondes sur chaque période de 5 secondes, ajustez **cpu.rt_runtime_us** sur **4000000** et **cpu.rt_period_us** sur **5000000**.

3.3. CPUACCT

Le sous-système **cpuacct** génère des rapports automatiques sur les ressources CPU utilisées par les tâches dans un groupe de contrôle, y compris les tâches dans les groupes enfants. Trois rapports sont disponibles :

cpuacct.stat

rapporte le nombre de cycles CPU (dans l'unité définie par **USER_HZ** sur le système) pris par les tâches dans ce groupe de contrôle et ses groupes enfant dans le mode utilisateur et dans le mode système (noyau).

cpuacct.usage

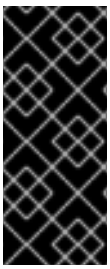
rapporte le temps de CPU total (en nanosecondes) pris par toutes les tâches dans ce groupe de contrôle (y compris les tâches situées plus bas dans la hiérarchie).

cpuacct.usage_percpu

rapporte le temps de CPU (en nanosecondes) pris sur chaque CPU par toutes les tâches présentes dans ce groupe de contrôle (y compris les tâches situées plus bas dans la hiérarchie).

3.4. CPUSET

Le sous-système **cpuset** assigne les CPU individuels et les noeuds de mémoire à des groupes de contrôle. Chaque cpuset peut être spécifié en fonction des paramètres suivants, chacun dans un *pseudo-fichier* différent du système de fichiers virtuel du groupe de contrôle :



IMPORTANT

Certains sous-systèmes ont des paramètres obligatoires à définir avant de pouvoir déplacer une tâche dans un groupe de contrôle qui utilise l'un de ces sous-systèmes. Par exemple, avant de déplacer une tâche dans un cgroup qui utilise le sous-système **cpuset**. les paramètres **cpuset.cpus** et **cpuset.mems** doivent être définis pour ce cgroup.

cpuset.cpus (obligatoire)

spécifie à quels CPU les tâches dans ce groupe de contrôle ont la permission d'accéder. Ceci est une liste séparée par des virgules sous le format ASCII, avec des tirets ("**-**") pour représenter l'étendue. Par exemple :

```
0-2,16
```

représente les CPUs 0, 1, 2, et 16.

cpuset.mems (obligatoire)

spécifie les noeuds de mémoire auxquels les tâches de ce groupe de contrôle auront la permission d'accéder. Ceci est une liste séparée par des virgules sous le format ASCII, avec des tirets ("**-**") pour repésenter l'étendue. Par exemple,

```
0-2,16
```

représente les noeuds de mémoire 0, 1, 2, et 16.

cpuset.memory_migrate

contient un indicateur (**0** ou **1**) spécifiant si une page en mémoire devrait migrer vers un nouveau noeud si les valeurs dans **cpuset.mems** changent. Par défaut, la migration de mémoire est désactivée (**0**) et les pages restent sur le noeud auquel elles ont été assignées à l'origine, même si celui-ci ne fait maintenant plus partie des noeuds spécifiés dans **cpuset.mems**. Si la migration est activée (**1**), le système migrera les pages vers les noeuds de mémoire faisant partie des nouveaux paramètres spécifiés par **cpuset.mems**, tout en maintenant leur position respective dans la mesure

du possible — par exemple, les pages sur le second noeud sur la liste spécifiée à l'origine par **cpuset.mems** seront allouées au second noeud sur la liste maintenant spécifiée par **cpuset.mems**, si cette position est disponible.

cpuset.cpu_exclusive

contient un indicateur (**0** ou **1**) spécifiant si des cpusets autres que celui-ci, que ses parents ou que ses enfants peuvent partager les CPUs spécifiés pour ce cpuset. Par défaut (**0**), les CPUs ne sont pas exclusivement alloués à un seul cpuset.

cpuset.mem_exclusive

contient un indicateur (**0** ou **1**) spécifiant si d'autres cpusets peuvent partager les noeuds de mémoire spécifiés pour ce cpuset. Par défaut (**0**), les noeuds de mémoire ne sont pas exclusivement alloués à un seul cpuset. Réserver des noeuds de mémoire pour une utilisation exclusive d'un seul cpuset ((**1**) est fonctionnellement la même chose qu'activer un hardwall de mémoire avec **cpuset.mem_hardwall**.

cpuset.mem_hardwall

contient un indicateur (**0** ou **1**) spécifiant si les allocations au noyau de pages mémoire et de données de tampon devraient être restreintes aux noeuds de mémoire spécifiés pour ce cpuset. Par défaut (**0**), les pages et les données de tampon sont partagées par les processus appartenant à de multiples utilisateurs. Avec un hardwall activé (**1**), la mémoire allouée aux tâches de chaque utilisateur peut être séparée de la mémoire allouée aux tâches de tous les autres utilisateurs.

cpuset.memory_pressure

un fichier en lecture seule qui contient une moyenne en cours d'exécution de la solicitation de mémoire (*memory pressure*), créée par les processus dans ce cpuset. La valeur dans ce pseudofichier est automatiquement mise à jour lorsque **cpuset.memory_pressure_enabled** est activé, sinon le pseudofichier contient la valeur **0**.

cpuset.memory_pressure_enabled

contient un indicateur (**0** ou **1**) spécifiant si le système doit calculer la solicitation de mémoire (*memory pressure*) créée par les processus dans ce groupe de contrôle. Les valeurs calculées sortent dans **cpuset.memory_pressure** et représentent le taux auquel les processus tentent de libérer la mémoire utilisée, ces valeurs sont rapportées en chiffres entiers et s'expliquent par le nombre de tentatives de récupération de mémoire par seconde, multiplié par 1000.

cpuset.memory_spread_page

contient un indicateur (**0** ou **1**) spécifiant si les tampons de systèmes de fichiers doivent être placés de manière régulièresur les noeuds de mémoire alloués à ce cpuset. Par défaut (**0**), aucune tentative de placer les pages de mémoire sur ces tampons de manière régulière n'est faite, et ceux-ci sont placés sur le noeud où le processus qui les a créés est en cours d'exécution.

cpuset.memory_spread_slab

contient un indicateur (**0** ou **1**) spécifiant si les caches slab du noyau pour opérations d'entrée/sortie doivet être placés de manière régulière sur le cpuset. Par défaut (**0**), aucune tentative de placer les caches slab du noyau de manière régulière n'est faite, et ceux-ci sont placés sur le noeud où le processus qui les a créés est en cours d'exécution.

cpuset.sched_load_balance

contient un indicateur (**0** ou **1**) spécifiant si le noyau va équilibrer les charges sur les CPU dans ce cpuset. Par défaut (**1**), le noyau équilibre les charges en déplaçant les processus des CPUs surchargés sur des CPUs moins solicités.

Remarquez cependant que définir cet indicateur dans un cgroup n'a aucun effet si l'équilibrage des charges est activé dans tout cgroup parent, car l'équilibrage des charges est déjà en cours à un niveau plus élevé. Ainsi, pour désactiver l'équilibrage des charges dans un cgroup, désactivez aussi l'équilibrage des charges dans chacun de ses parents dans la hiérarchie. Dans ce cas, vous devriez aussi vous demander si l'équilibrage des charges devrait être activé sur les autres relations de même parenté du groupe de contrôle en question.

cpuset.sched_relax_domain_level

contient un entier entre **-1** et une petite valeur positive, qui représente la largeur de l'étendue des CPUs sur laquelle le noyau va essayer d'équilibrer les charges. Cette valeur n'a aucun sens si **cpuset.sched_load_balance** est désactivé.

L'effet précis de cette valeur varie en fonction de l'architecture du système, mais les valeurs suivantes sont typiques :

Valeurs de cpuset.sched_relax_domain_level

Valeur	Effet
-1	Utilise la valeur par défaut du système pour l'équilibrage des charges
0	N'effectue pas d'équilibrage des charges immédiat ; équilibre les charges de manière périodique uniquement
1	Équilibre immédiatement les charges sur les threads du même coeur
2	Équilibre immédiatement les charges sur les coeurs dans le même paquetage
3	Équilibre immédiatement les charges sur les CPUs du même noeud ou de la même lame
4	Équilibre immédiatement les charges sur plusieurs CPUs sur des architectures à accès non uniforme à la mémoire (NUMA)
5	Équilibre immédiatement les charges sur tous les CPUs sur architectures avec NUMA

3.5. PÉRIPHÉRIQUES

Le sous-système **devices** autorise ou refuse l'accès des tâches aux périphériques dans un groupe de contrôle.

IMPORTANT

Le sous-système Device Whitelist (**devices**) est considéré comme un aperçu technologique dans Red Hat Enterprise Linux 6.

Les fonctionnalités des *Aperçus technologiques* ne sont actuellement pas prises en charge par les services d'abonnement Red Hat Enterprise Linux 6, ils pourraient ne pas être complets et ne sont généralement pas convenables pour une utilisation en milieu de production. Cependant, Red Hat inclut ces fonctionnalités dans le système d'exploitation au service du client et afin de les exposer à une plus large audience. Ces fonctionnalités peuvent se révéler utiles dans un environnement qui n'est pas un environnement de production. Il vous est aussi possible de soumettre des commentaires ou d'envoyer des suggestions sur la fonction d'un aperçu technologique avant que celle-ci ne soit complètement prise en charge.

devices.allow

spécifie à quels périphériques les tâches d'un groupe de contrôle peuvent avoir accès. Chaque entrée possède quatre champs : *type*, *majeur*, *mineur* et *accès*. Les valeurs utilisées dans les champs *type*, *majeur* et *mineur* correspondent au type de périphérique et aux numéros de noeuds spécifiés dans *Périphériques alloués Linux*, aussi connu sous le nom de *Liste des périphériques Linux* et disponible depuis http://www.kernel.org/doc/Documentation/devices.txt.

type

type peut avoir l'une des trois valeurs suivantes :

- **a** — s'applique à tous les périphériques, *périphérique de type caractère* et *périphériques blocs*
- **b** — spécifie un périphérique bloc
- **c** — spécifie un périphérique de type caractère

majeur, mineur

majeur et *mineur* sont des numéros de noeuds de périphériques spécifiés par *Périphériques alloués Linux*. Les numéros majeurs et mineurs sont séparés par deux points. Par exemple, **8** est le numéro majeur spécifiant les lecteurs de disques SCSI, et le numéro mineur **1** spécifie la première partition du premier lecteur de disque SCSI ; ainsi, **8:1** spécifie cette partition en détails, correspondant à l'emplacement sur le système de fichiers de **/dev/sda1**.

***** peut remplacer tous les noeuds de périphériques majeurs ou mineurs. Par exemple, **9:*** (tous les périphériques RAID), ou même ***:*** (tous les périphériques).

accès

accès est une séquence d'une ou plusieurs des lettres suivantes :

- **r** — autorise les tâches à lire le périphérique spécifié
- **w** — autorise les tâches à écrire sur le périphérique spécifié
- **m** — autorise les tâches à créer des fichiers de périphériques qui n'existent pas encore

Par exemple, lorsque l'*accès* est spécifié en tant que **r**, les tâches peuvent uniquement lire le périphérique spécifié, mais lorsque l'*accès* est spécifié en tant que **rw**, les tâches peuvent lire et écrire sur le périphérique.

devices.deny

spécifie à quels périphériques les tâches dans un groupe de contrôle ne peuvent pas avoir accès. La syntaxe des entrées est identique à celle de **devices.allow**.

devices.list

rapporte les périphériques pour lesquels les contrôles d'accès ont été définis pour des tâches dans ce groupe de contrôle.

3.6. FREEZER

Le sous-système **freezer** suspend ou réactive les tâches dans un groupe de contrôle.

freezer.state

freezer.state possède trois valeurs possibles :

- **FROZEN** — les tâches dans le groupe de contrôle sont suspendues.
- **FREEZING** — le système est en train de suspendre les tâches dans le groupe de contrôle.
- **THAWED** — les tâches dans le groupe de contrôle sont réactivées.

Pour suspendre un processus spécifique :

1. Déplacez ce processus dans un groupe de contrôle se trouvant dans une hiérarchie à laquelle le sous-système **freezer** est attaché.
2. Gelez ce groupe de contrôle en particulier pour suspendre le processus qu'il contient.

Il n'est pas possible de déplacer un processus dans un groupe de contrôle suspendu (frozen).

Remarquez que les valeurs **FROZEN** et **THAWED** peuvent être écrites sur **freezer.state**, alors que **FREEZING** peut seulement être lu.

3.7. MÉMOIRE

Le sous-système **mémoire** génère des rapports automatiques sur les ressources mémoire utilisées par les tâches dans un groupe de contrôle. Il définit aussi les limites d'utilisation de mémoire de ces tâches :

memory.stat

rapporte un large éventail de statistiques de mémoire, comme décrit dans le tableau suivant :

Tableau 3.1. Les valeurs rapportées par memory.stat

Statistique	Description
cache	cache de la page, inclut **tmpfs** (**shmem**), en octets
rss	caches swap et anonyme, *n'inclut pas* **tmpfs** (**shmem**), en octets
mapped_file	taille des fichiers mappé en mémoire, inclut **tmpfs** (**shmem**), en octets
pgpgin	nombre de pages chargées en mémoire
pgpgout	nombre de pages renvoyées de la mémoire
swap	usage swap, en octets
active_anon	caches swap et anonyme de la liste des LRU (dernier récemment utilisé) actifs, inclut **tmpfs** (**shmem**), en octets
inactive_anon	caches swap et anonyme de la liste des LRU (dernier récemment utilisé) inactifs, inclut **tmpfs** (**shmem**), en octets
active_file	mémoire sauvegardée sur fichier de la liste des LRU actifs, en octets
inactive_file	mémoire sauvegardée sur fichier de la liste des LRU inactifs, en octets
unevictable	mémoire ne pouvant pas être récupérée, en octets
hierarchical_memory_limit	limite de la mémoire pour la hiérarchie contenant le groupe de contrôle **memory**, en octets
hierarchical_memsw_limit	limite de la mémoire plus le swap pour la hiérarchie contenant le groupe de contrôle **memory**, en octets

En outre, à l'exception de **hierarchical_memory_limit** et de **hierarchical_memsw_limit**, tous les fichiers possèdent un préfixe **total_** qui effectue un rapport non seulement sur le groupe de contrôle, mais aussi sur ses enfants. Par exemple, **swap** rapporte l'utilisation du swap d'un groupe de contrôle et **total_swap** rapporte l'utilisation totale du swap du groupe de contrôle et de tous ses groupes enfants.

Lorsque vous interprétez les valeurs rapportées par **memory.stat**, remarquez comme les diverses statistiques sont reliées entres elles :

- **active_anon** + **inactive_anon** = mémoire anonyme + cache du fichier de **tmpfs** + cache du swap

  Ainsi, **active_anon** + **inactive_anon** ≠ **rss**, car **rss** n'inclut pas **tmpfs**.

- **active_file** + **inactive_file** = cache - taille de **tmpfs**

memory.usage_in_bytes

rapporte la totalité de l'utilisation de mémoire actuelle par les processus dans le groupe de contrôle (en octets).

memory.memsw.usage_in_bytes

rapporte la somme de l'utilisation actuelle de la mémoire et de l'espace swap utilisé par les processus dans le groupe de contrôle (en octets).

memory.max_usage_in_bytes

rapporte le montant maximum de mémoire utilisée par les processus dans le groupe de contrôle (en octets).

memory.memsw.max_usage_in_bytes

rapporte le montant maximum de mémoire et d'espace swap utilisé par les processus dans le groupe de contrôle (en octets).

memory.limit_in_bytes

définit le montant maximum de mémoire utilisateur (y compris le cache du fichier). Si aucune unité n'est saisie, la valeur sera interprétée en octets. Il est cependant possible d'utiliser des suffixes afin de représenter de plus grandes unités — **k** ou **K** pour des kilo-octets, **m** ou **M** pour des mégaoctets, et **g** ou **G** pour des gigaoctets.

L'utilisation de **memory.limit_in_bytes** pour limiter le groupe de contrôle racine est impossible ; l'application de valeurs est uniquement possible sur des groupes se trouvant plus bas dans la hiérarchie.

Écrire **-1** sur **memory.limit_in_bytes** pour supprimer toutes les limites existantes.

memory.memsw.limit_in_bytes

définit le montant maximum de la somme de l'utilisation de la mémoire et du swap. Si aucune unité n'est saisie, la valeur sera interprétée en octets. Il est cependant possible d'utiliser des suffixes afin de représenter de plus grandes unités — **k** ou **K** pour des kilo-octets, **m** ou **M** pour des mégaoctets, et **g** ou **G** pour des gigaoctets.

L'utilisation de **memory.memsw.limit_in_bytes** pour limiter le groupe de contrôle racine est impossible ; l'application de valeurs est uniquement possible sur des groupes se trouvant plus bas dans la hiérarchie.

Écrire **-1** sur **memory.memsw.limit_in_bytes** pour supprimer toutes les limites existantes.

memory.failcnt

rapporte le nombre de fois que la limite de mémoire a atteint la valeur définie dans **memory.limit_in_bytes**.

memory.memsw.failcnt

rapporte le nombre de fois que la limite de la somme de la mémoire et de l'espace swap a atteint la valeur définie dans **memory.memsw.limit_in_bytes**.

memory.force_empty

vide la mémoire de toutes les pages utilisées par des tâches dans ce groupe de contrôle lorsque

défini sur **0**. Cette interface peut uniquement être utilisée lorsque le groupe de contrôle ne possède aucune tâche. Si la mémoire ne peut pas être libérée, celle-ci sera déplacée sur un groupe de contrôle parent si possible. Utilisez **memory.force_empty** avant de supprimer un groupe de contrôle afin d'éviter de déplacer les caches des pages hors d'usage sur son groupe de contrôle parent.

memory.swappiness

définit la tendance du noyau à déloger la mémoire de processus utilisée par les tâches dans ce groupe de contrôle plutôt que de réclamer des pages depuis le cache de page. Il s'agit de la même tendance et du même calcul que ce qui figure dans **/proc/sys/vm/swappiness** pour le système en tant que tout. La valeur par défaut est **60**. Des valeurs plus petites diminueront la tendance à déloger la mémoire de processus. Des valeurs plus élevées que **60** augmenteront la tendance du noyau à déloger la mémoire de processus, et les valeurs supérieures à **100** permettront au noyau de déloger des pages faisant partie de l'espace adresse pour ce groupe de contrôle.

Remarquez que la valeur **0** n'empêchera pas la mémoire de processus d'être délogée; celui-ci pourra tout de même se produire lorsqu'il y a un manque de mémoire système car la logique de gestion de la mémoire virtuelle globale ne lit pas la valeur du groupe de contrôle. Pour complètement verrouiller les pages, veuillez utiliser **mlock()** au lieu des groupes de contrôle.

Il n'est pas possible de modifier le "swappiness" des groupes suivants :

- le groupe de contrôle racine, qui utilise le "swappiness" installé dans **/proc/sys/vm/swappiness**.
- un groupe de contrôle possédant des groupes enfants.

memory.use_hierarchy

contient un indicateur (**0** ou **1**) spécifiant si la mémoire utilisée devrait être prise en compte sur une hiérarchie de groupes de contrôle. Si activé (**1**), le sous-système de la mémoire récupèrera la mémoire de tous les processus enfants excédant la limite de mémoire. Par défaut (**0**), le sous-système ne récupère pas la mémoire d'une tâche enfant.

3.8. NET_CLS

Le sous-système **net_cls** repère les paquets réseau avec un identifiant de classe (classid) qui permet au contrôleur de trafic Linux (**tc**) d'identifier les paquets provenant d'un groupe de contrôle particulier. Le contrôleur de trafic peut être configuré de manière à assigner différentes priorités à des paquets provenant de différents groupes de contrôle.

net_cls.classid

net_cls.classid contient une valeur seule sous format hexadécimal indiquant un *descripteur* (ou *handle*) de contrôle du trafic. Par exemple, **0x100001** représente le handle conventionnellement écrit sous le format utilisé par iproute2 : **10:1**.

Le format de ces descripteurs (handles) est comme suit : **0x*AAAABBBB***, où *AAAA* est le numéro majeur en hexadécimal et où *BBBB* est le numéro mineur en hexadécimal. Vous pouvez ainsi omettre d'inclure les zéros du début : **0x10001** est la même chose que **0x00010001**, et représente **1:1**.

Reportez-vous à la page man de **tc** afin d'apprendre comment configurer le contrôleur de trafic pour qu'il utilise les handles que **net_cls** ajoute aux paquets réseau.

3.9. NS

Le sous-système **ns** fournit une manière de grouper les processus dans des *espaces de noms* (*namespaces*) séparés. Dans un espace de nom particulier, les processus peuvent interagir ensemble mais sont isolés des processus exécutés dans d'autres espaces de noms. Ces autres espaces de noms sont parfois appelés des *conteneurs* (*containers*) lors de leur utilisation pour une virtualisation au niveau du système d'exploitation.

3.10. RESSOURCES SUPPLÉMENTAIRES

Documentation sur les noyaux spécifiques aux sous-systèmes

Tous les fichiers suivants se trouvent sous le répertoire **/usr/share/doc/kernel-doc-<*kernel_version*>/Documentation/cgroups/** (fournit par le paquetage kernel-doc).

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

ANNEXE A. HISTORIQUE DES RÉVISIONS

Version 1-2.33.400 **2013-10-31** **Rüdiger Landmann**

Rebuild with publican 4.0.0

Version 1-2.33 **July 24 2012** **Ruediger Landmann**

Rebuild for Publican 3.0

Version 1.0-5 **Thu May 19 2011** **Martin Prpič**

Mise à disponibilité générale du *Guide de gestion des ressources* de Red Hat Enterprise Linux 6.1.

Version 1.0-4 **Tue Mar 1 2011** **Martin Prpič**

Correction de multiples exemples — BZ#667623, BZ#667676, BZ#667699

Clarification de la commande cgclear— BZ#577101

Clarification de la commande lssubsystem — BZ#678517

Gel d'un processus — BZ# 677548

Version 1.0-3 **Wed Nov 17 2010** **Rüdiger Landmann**

Exemple de remontage correct — BZ#612805

Version 1.0-2 **Thu Nov 11 2010** **Rüdiger Landmann**

Suppression des instructions des commentaires antérieurs à la publication

Version 1.0-1 **Wed Nov 10 2010** **Rüdiger Landmann**

Corrections de QE — BZ#581702 et BZ#612805

Version 1.0-0 **Tue Nov 9 2010** **Rüdiger Landmann**

Version complète pour la disponibilité générale