Introducción a los grupos de control (cgroups) de Linux

Los grupos de control, *cgroups*, de Linux son una excelente herramienta para controlar la asignación de recursos a los procesos. Dado un proceso los permisos tradicionales (o las ACLs, o los permisos MAC como SELinux) limitan a qué se accede y qué operaciones se permiten, los límites (/**etc**/**security**/**limits.conf**) acotan la máxima asignación de recursos, y el planificador intentará asignar recursos de manera equitativa pero influenciado por los valores de prioridad y nice (**ionice** cuando se trata de regular la entrada/salida). Pero todas estas herramientas que son muy útiles no permiten que el administrador especifique con detalle cómo se deben asignar los recursos entre las tareas, sobre todo cuando, como ocurre a menudo, todas quieren todos los recursos disponibles. Y aún así no se quiere consentir que porque algunos procesos acaparen tantos recursos otros se queden con tan pocos que no puedan realizar su trabajo, o lo hagan tan lentamente que en la práctica se haya producido una denegación de servicio.

Los grupos de control permiten definir jerarquías en las que se agrupan los procesos de manera que un administrador puede definir con gran detalle la manera en la que se asignan los recursos (no solo tiempo de atención de CPU, sino también I/O y memoria principal) o llevar la contabilidad de los mismos.

Un ejemplo en el que los cgroups son de ayuda

Cuando se reparte el tiempo de atención de CPU entre los procesos que se quieren ejecutar se intenta dar un trato equitativo a tareas con la misma prioridad. En Linux se puede cambiar el valor nice de un proceso para hacerlo más prioritario (al disminuir el valor) o menos prioritario (al aumentar su valor).

Suponiendo una sola CPU y dos procesos, se podrá actuar sobre el valor nice para hacer que el planificador dedique más atención a uno u otro.

En la mayoría de casos reales nos encontramos con servicios, por ejemplo Apache, que lanzan subprocesos de manera dinámica. Cuando se llama a fork, el proceso hijo hereda el valor de nice del proceso padre. Así que si por ejemplo, se ha lanzado Apache con un valor de nice +5 todos sus procesos hijos lo heredarán. Pero como ya sabrá Apache mantiene un *pool* de procesos hijos listos para atender a los clientes. Cuando hay pocas solicitudes su número se reduce y a medida que aumenta la demanda su número aumenta hasta llegar al máximo configurado.

En este ejemplo el problema viene dado porque el número total de procesos que utiliza Apache es variable, y respecto a cualquier otro servicio que se esté ejecutando en la máquina eso va a hacer que la ración de atención de CPU disponible sea variable.

Se puede ver claramente que:

- Si apache tiene 9 procesos y otro servicio 1. Cuando todos se quieran ejecutar, y suponiendo la misma prioridad al otro servicio le tocará un 10% de CPU.
- Si apache tiene 99 procesos y otro servicio 1. Cuando todos se quieran ejecutar, y suponiendo la misma prioridad al otro servicio le tocará un 1% de CPU.

Así es que la atención que recibe el segundo servicio dependerá del número de

procesos que lance Apache, por mucho que todos tengan siempre el mismo valor de nice.

Gracias a los grupos de control se podrían crear dos categorias: apache y otro. Agrupando a Apache con todos sus procesos hijos en la primera y dejando la segunda para el otro servicio. Una vez que esto esté hecho se podrá actuar sobre estos grupos indicando directamente la porción de atención de CPU asignada. Por ejemplo 50% para cada uno, y esta proporción se mantendrá independientemente del número de procesos que tenga cada uno de los servicios. Es decir, Apache obtendrá el 50% de la CPU cuente con 9 o 99 procesos. Y el otro servicio, que en el ejemplo solo tenía un proceso, el 50% restante.

Trabajando con cgroups

Dependiendo de la distribución de GNU/Linux los cgroups se utilizan de serie, como ocurre en Fedora donde systemd se integra con los cgroups agrupando todos los servicios que se lanzan. En otras, como Ubuntu 12.04 LTS, no se utilizan por defecto pero se pueden activar sin ningún problema.

Los cgroups se pueden manipular a través de un sistema de archivos virtual. A tal efecto, en **/sys/fs/cgroup** se encuentra el directorio que contendrá los puntos de montaje de las diferentes jerarquías (se pueden montar varios cgroups) que inicialmente está vacío.

Para utilizar la característica que permite controlar la asignación de CPUs a los procesos podemos hacer esto:

```
mount -t tmpfs -o size=5M cgroup_root /sys/fs/cgroup
mkdir /sys/fs/cgroup/cpuset
mount -t cgroup -ocpuset cpuset /sys/fs/cgroup/cpuset
```

Con lo que:

1. Hemos montado un pequeño (5M) sistema de ficheros **tmpfs** en **/sys/fs/cgroup** para poder crear directorios, como cpuset, que servirán de punto de montaje a las jerarquías de cgroups. Cabe destacar que '**cgroup_root**' es opcional y no se trata más que de un nombre para que al hacer **df** o **mount** contemos con la etiqueta adecuada.
2. Hemos creado un directorio **cpuset**, que podría tener cualquier otro nombre.
3. Hemos montado un sistema de archivos **cgroup**, para el subsistema **cpuset** (que controla la asignación de CPUs) en dicho directorio. La opción **-ocpuset** indica el subsistema de cgroup que se va a montar, el parámetro **cpuset** del mount no es más que una etiqueta como en el primer caso.

Una vez realizado el montaje se puede comprobar que **/sys/fs/cgroup/cpuset** contiene ficheros que nos permiten interactuar con el grupo de control:

```
root@sombragris:/sys/fs/cgroup/cpuset# ls
cgroup.clone_children   cpuset.memory_pressure_enabled
cgroup.event_control    cpuset.memory_spread_page
cgroup.procs            cpuset.memory_spread_slab
cpuset.cpu_exclusive    cpuset.mems
cpuset.cpus             cpuset.sched_load_balance
cpuset.mem_exclusive    cpuset.sched_relax_domain_level
cpuset.mem_hardwall     notify_on_release
```

```
cpuset.memory_migrate   release_agent
cpuset.memory_pressure  tasks
root@sombragris:/sys/fs/cgroup/cpuset#
```

De estos ficheros hay algunos que forman parte de cualquier cgroup, como **tasks** y **cgroup.procs** y otros que son específicos del subsistema **cpuset** que estamos utilizando.

De los comunes, de momento nos vamos a fijar en:

tasks

> Lista de tareas que están asociadas con el cgroup. Al añadir un PID a este fichero se asocia una nueva tarea.

cgroup.procs

> Lista de TGIDs (*Thread Group IDs*) en el cgroup.

Respecto a los propios del subsistema, o controlador de recursos, cpuset nos vamos a fijar en:

cpuset.cpus

> Especifica la lista de CPUs utilizables por las tareas del grupo de control. Debe inicializarse antes de añadir tareas al grupo.

cpuset.mems

> Especifica la lista de nodos de memoria utilizables por las tareas de grupo de control. Debe inicializarse antes de añadir tareas al grupo.

Así, en nuestro recién estrenado sistema de archivos /**sys**/**fs**/**cgroup**/**cpuset** nos encontramos con que:

- **tasks** contiene la lista de todos los PIDs en uso
- **cgroup.procs** contiene la lista de todos los TGIDs en uso
- **cpuset.cpus** contiene la lista de núcleos utilizables (0-1, en una máquina con dos núcleos)
- **cpuset.mems** contiene la lista de nodos de memoria utilizables (0, en la máquina de ejemplo)

Nos encontramos con que todas las tareas de la máquina están catalogadas en el único cgroup que existe, y dicho grupo puede utilizar los dos núcleos y el único banco de memoria disponible. Pero cuando se trata de controlar recursos, lo que se debe hacer es subgrupos de control formando una jerarquía (el actual es el padre de todos). De manera que se catalogarán con más detalle las tareas en los subgrupos de control donde recibirán diferente trato.

Crear un subgrupo de control es tan sencillo como crear un subdirectorio, y eliminarlo es tan sencillo como borrar dicho directorio con **rmdir** (no sirve **rm -r** y el grupo de control a eliminar no debe estar controlando a ninguna tarea).

Así, crear dos subgrupos de control y asignar uno de los núcleos a cada uno de ellos es algo tan sencillo como:

```
root@sombragris:/sys/fs/cgroup/cpuset# mkdir grupo-A grupo-B
```

```
root@sombragris:/sys/fs/cgroup/cpuset# echo 0 >grupo-A/cpuset.cpus
root@sombragris:/sys/fs/cgroup/cpuset# echo 1 >grupo-B/cpuset.cpus
root@sombragris:/sys/fs/cgroup/cpuset# echo 0 >grupo-A/cpuset.mems
root@sombragris:/sys/fs/cgroup/cpuset# echo 0 >grupo-B/cpuset.mems
```

En este caso:

1. Se han creado dos grupos de control: **grupo-A** y **grupo-B**
2. Se ha indicado que **grupo-A** podrá hacer uso del núcleo **0** y **grupo-B** podrá hacer uso del núcleo **1**
3. Se ha indicado que ambos podrán hacer uso del nodo de memoria **0** que es el único disponible

Una vez hecho esto solo nos falta catalogar algunas tareas dentro de su grupo de control al escribir PIDs en el fichero **tasks** correspondiente de uno en uno. Con esto se comprobará que dada una jerarquía de cgroups, un proceso solo puede estar catalogado en un grupo. De manera que al añadir su PID al fichero **grupo-A/tasks** desaparecerá del grupo raíz que lo contenía.

Si el proceso que se ha categorizado en el subgrupo **grupo-A** hiciese uso de **fork** para lanzar nuevos procesos, estos heredarían la pertenencia al grupo en el que está el proceso padre. Así se agrupa tanto un proceso como aquellos hijos que pueda lanzar.

Para comprobarlo abriremos dos nuevos shells. En cada uno puede comprobar el PID del proceso bash mediante la orden '**echo $$**'. En este ejemplo ha resultado que los dos nuevos bash corresponden a los procesos **3435** y **3492**.

Se puede comprobar que dichos PIDs forman parte del fichero tasks del grupo de control raíz, y que al moverlos a los subgrupos (uno al A y otro al B) desaparecen del mismo.

```
root@sombragris:/sys/fs/cgroup/cpuset# cat tasks | grep -E "3435|3492"
3435
3492
root@sombragris:/sys/fs/cgroup/cpuset# echo 3435 >grupo-A/tasks
root@sombragris:/sys/fs/cgroup/cpuset# echo 3492 >grupo-B/tasks
root@sombragris:/sys/fs/cgroup/cpuset# cat tasks | grep -E "3435|3492"
root@sombragris:/sys/fs/cgroup/cpuset#
```

A partir de ahora todos los procesos que se lancen desde uno de estos shells quedarán catalogados en el grupo de control correspondiente. Por ejemplo, al lanzar cuatro **consume_cpu** (que únicamente consume cpu en un bucle infinito) desde el shell que tiene por PID **3435** los procesos quedarán atrapados en el **grupo-A** y únicamente podrán hacer uso del procesador **0**.

```
vcarceler@sombragris:~$ ~/bin/consume_cpu &
[1] 3621
vcarceler@sombragris:~$ ~/bin/consume_cpu &
[2] 3622
vcarceler@sombragris:~$ ~/bin/consume_cpu &
[3] 3623
vcarceler@sombragris:~$ ~/bin/consume_cpu &
[4] 3624
vcarceler@sombragris:~$ cat /sys/fs/cgroup/cpuset/grupo-A/tasks
3435
3621
3622
3623
3624
3654
```

```
vcarceler@sombragris:~$
```

El último proceso de la lista, **3654**, corresponde al propio **cat** mientras se estaba ejecutando. De manera que al comprobar el uso de CPU mediante **top** nos encontramos con un núcleo muy ocupado, otro que no trabaja y cada **consume_cpu** con un 25% de atención del núcleo que trabaja.

```
Tasks: 174 total,   9 running, 165 sleeping,   0 stopped,   0 zombie
Cpu0  : 99.7%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.3%si,  0.0%st
Cpu1  :  5.8%us,  3.1%sy,  0.0%ni, 91.2%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   4041412k total,  2203684k used,  1837728k free,      976k buffers
Swap:  4921340k total,        0k used,  4921340k free,  1194676k cached

 PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3623 vcarcele  20   0  4156  356  276 R   25  0.0   1:09.66 consume_cpu
 3621 vcarcele  20   0  4156  356  276 R   25  0.0   1:11.30 consume_cpu
 3622 vcarcele  20   0  4156  356  276 R   25  0.0   1:10.11 consume_cpu
 3624 vcarcele  20   0  4156  352  276 R   25  0.0   1:09.35 consume_cpu
```

Llegados a este punto bastará con escribir el PID de uno de los cuatro procesos en el fichero **tasks** de otro cgroup para migrarlo y que tenga un mejor trato. Por ejemplo podemos moverlo al **grupo-B**.

```
echo 3621 >/sys/fs/cgroup/cpuset/grupo-B/tasks
```

y comprobar con top el resultado:

```
Tasks: 175 total,   7 running, 168 sleeping,   0 stopped,   0 zombie
Cpu0  : 98.3%us,  1.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.7%si,  0.0%st
Cpu1  : 98.0%us,  2.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   4041412k total,  2333092k used,  1708320k free,      976k buffers
Swap:  4921340k total,        0k used,  4921340k free,  1277012k cached

 PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3621 vcarcele  20   0  4156  356  276 R   87  0.0   4:02.21 consume_cpu
 3622 vcarcele  20   0  4156  356  276 R   32  0.0   3:38.79 consume_cpu
 3623 vcarcele  20   0  4156  356  276 R   32  0.0   3:38.34 consume_cpu
 3624 vcarcele  20   0  4156  352  276 R   31  0.0   3:38.03 consume_cpu
```

Ahora los dos núcleos trabajan por igual, uno se ocupa de un solo **consume_cpu** y el otro de los tres restantes. Tenga en cuenta que en la máquina se están ejecutando otras tareas.

Otros subsistemas o controladores de recursos

En la terminología de cgroups, **cpuset** es un subsistema o controlador de recursos. Pero hay más subsistemas disponibles:

blkio

> Controla y monitoriza la entrada/salida con los dispositivos de bloques. Se puede regular el ancho de banda proporcionalmente o con un techo.

cpu

> Actúa sobre el planificador para determinar la proporción de uso de CPU. Por ejemplo se puede especificar que un grupo de control disponga del 80% de la CPU y otros dos del 15% y 5% restante.

cpuacct

No influye en la asignación de recursos, únicamente lleva la cuenta del tiempo de CPU consumido por las tareas del grupo de control y sus subgrupos. Su finalidad es contabilizar cómo se reparte el tiempo de CPU.

cpuset

Permite asignar núcleos y nodos de memoria a grupos de control. Si se hace de forma exclusiva (**cpuset.cpu_exclusive** puesto a 1) entonces no se permitirá compartir dicho núcleo con ningún grupo hermano, aunque sí que se podrá asignar a subgrupos de control.

devices

Sirve para permitir o denegar el acceso a dispositivos concretos para aquellas tareas que estén en el grupo de control.

freezer

Suspende o reactiva tareas.

memory

Fija límites a la memoria consumida por las tareas del cgroup y genera informes sobre dicho consumo.

net_cls

Etiqueta los paquetes generados por procesos del grupo de control, de manera que el controlador de tráfico de Linux (**tc**) pueda identificarlos y tratarlos de manera adecuada.

net_prio

Cambia la prioridad del tráfico de red por interfaz.

ns

Agrupa los procesos en diferentes espacios de nombres. Los procesos únicamente pueden interactuar con aquellos que están en el mismo espacio de nombres, es de utilidad en la virtualización a nivel de sistema operativo.

perf_event

Hace que las tareas del grupo de control se puedan monitorizar con la herramienta **perf**.

Más información:

- Kernel documentation: cgroups
- Wikipedia: cgroups
- RHEL 6: Resource Management Guide

Santa Coloma de Gramenet
Avinguda Anselm de Riu 10
Tn: 93 391 61 11
Fax: 93 392 44 07
E-mail: iespuigcastellar@xtec.cat

Informació general
Documents de referència
Biblioteca
Revista
Secretaria
IOC
AMPA

Correu electrònic, agenda
Moodle
El blog del Puig
Gestió de faltes d'assistència
Saga - Portal de centre - Esfer@
ownCloud

Administració, Biologia i Geologia, Català, Clàssiques, Dibuix, Economia, Eduació Física, Filosofia, Física i Química, História, Idiomes, Informàtica, Llengua Castellana, Matemàtiques, Música, Psicopedagogia, Religió, Tecnologia

Sota el cel del puig.
Junio negro.
Sendero Cósmico.
¿A usted qué le importa?.