
Subject: Re: [RFC] Control Groups Roadmap ideas
Posted by [Balbir Singh](#) on Mon, 14 Apr 2008 14:31:02 GMT
[View Forum Message](#) <> [Reply to Message](#)

Serge E. Hallyn wrote:

> Quoting Balbir Singh (balbir@linux.vnet.ibm.com):
>> On Fri, Apr 11, 2008 at 8:18 PM, Serge E. Hallyn <serue@us.ibm.com> wrote:
>>> Quoting Paul Menage (menage@google.com):
>>> > This is a list of some of the sub-projects that I'm planning for
>>> > Control Groups, or that I know others are planning on or working on.
>>> > Any comments or suggestions are welcome.
>>> >
>>> >
>>> > 1) Stateless subsystems
>>> > -----
>>> >
>>> > This was motivated by the recent "freezer" subsystem proposal, which
>>> > included a facility for sending signals to all members of a cgroup.
>>> > This wasn't specifically freezer-related, and wasn't even something
>>> > that needed particular per-cgroup state - its only state is that set
>>> > of processes, which is already tracked by cgroups. So it could
>>> > theoretically be mounted on multiple hierarchies at once, and wouldn't
>>> > need an entry in the css_set array.
>>> >
>>> > This would require a few internal plumbing changes in cgroups, in particular:
>>> >
>>> > - hashing css_set objects based on their cgroups rather than their css pointers
>>> > - allowing stateless subsystems to be in multiple hierarchies
>>> > - changing the way hierarchy ids are calculated - simply ORing
>>> > together the subsystem would no longer work since that could result in
>>> > duplicates
>>> >
>>> > 2) More flexible binding/unbinding/rebinding
>>> > -----
>>> >
>>> > Currently you can only add/remove subsystems to a hierarchy when it
>>> > has just a single (root) cgroup. This is a bit inflexible, so I'm
>>> > planning to support:
>>> >
>>> > - adding a subsystem to an existing hierarchy by automatically
>>> > creating a subsys state object for the new subsystem for each existing
>>> > cgroup in the hierarchy and doing the appropriate
>>> > can_attach()/attach_tasks() callbacks for all tasks in the system
>>> >
>>> > - removing a subsystem from an existing hierarchy by moving all tasks
>>> > to that subsystem's root cgroup and destroying the child subsystem
>>> > state objects
>>> >

```

>>> > - merging two existing hierarchies that have identical cgroup trees
>>> >
>>> > - (maybe) splitting one hierarchy into two separate hierarchies
>>> >
>>> > Whether all these operations should be forced through the mount()
>>> > system call, or whether they should be done via operations on cgroup
>>> > control files, is something I've not figured out yet.
>>>
>>> I'm tempted to ask what the use case is for this (I assume you have one,
>>> you don't generally introduce features for no good reason), but it
>>> doesn't sound like this would have any performance effect on the general
>>> case, so it sounds good.
>>>
>>> I'd stick with mount semantics. Just
>>>     mount -t cgroup -o remount,devices,cpu none /devwh"
>>> should handle all cases, no?
>>>
>>>
>>>
>>> > 3) Subsystem dependencies
>>> > -----
>>> >
>>> > This would be a fairly simple change, essentially allowing one
>>> > subsystem to require that it only be mounted on a hierarchy when some
>>> > other subsystem was also present. The implementation would probably be
>>> > a callback that allows a subsystem to confirm whether it's prepared to
>>> > be included in a proposed hierarchy containing a specified subsystem
>>> > bitmask; it would be able to prevent the hierarchy from being created
>>> > by giving an error return. An example of a use for this would be a
>>> > swap subsystem that is mostly independent of the memory controller,
>>> > but uses the page-ownership tracking of the memory controller to
>>> > determine which cgroup to charge swap pages to. Hence it would require
>>> > that it only be mounted on a hierarchy that also included a memory
>>> > controller. The memory controller would make no such requirement by
>>> > itself, so could be used on its own without the swap controller.
>>> >
>>> >
>>> > 4) Subsystem Inheritance
>>> > -----
>>> >
>>> > This is an idea that I've been kicking around for a while trying to
>>> > figure out whether its usefulness is worth the in-kernel complexity,
>>> > versus doing it in userspace. It comes from the idea that although
>>> > cgroups supports multiple hierarchies so that different subsystems can
>>> > see different task groupings, one of the more common uses of this is
>>> > (I believe) to support a setup where say we have separate groups A, B
>>> > and C for one resource X, but for resource Y we want a group
>>> > consisting of A+B+C. E.g. we want individual CPU limits for A, B and

```

```

>>> > C, but for disk I/O we want them all to share a common limit. This can
>>> > be done from userspace by mounting two hierarchies, one for CPU and
>>> > one for disk I/O, and creating appropriate groupings, but it could
>>> > also be done in the kernel as follows:
>>> >
>>> > - each subsystem "foo" would have a "foo.inherit" file provided by
>>> > (and handled by) cgroups in each group directory
>>> >
>>> > - setting the foo.inherit flag (i.e. writing 1 to it) would cause
>>> > tasks in that cgroup to share the "foo" subsystem state with the
>>> > parent cgroup
>>> >
>>> > - from the subsystem's point of view, it would only need to worry
>>> > about its own foo_cgroup objects and which task was associated with
>>> > each object; the subsystem wouldn't need to care about which tasks
>>> > were part of each cgroup, and which cgroups were sharing state; that
>>> > would all be taken care of by the cgroup framework
>>> >
>>> > I've mentioned this a couple of times on the containers list as part
>>> > of other random discussions; at one point Serge Hallyn expressed some
>>> > interest but there's not been much noise about it either way. I
>>> > figured I'd include it on this list anyway to see what people think of
>>> > it.
>>>
>>> I guess I'm hoping that if libcg goes well then a userspace daemon can
>>> do all we need. Of course the use case I envision is having a container
>>> which is locked to some amount of ram, wherein the container admin wants
>>> to lock some daemon to a subset of that ram. If the host admin lets the
>>> container admin edit a config file (or talk to a daemon through some
>>> sock designated for the container) that will only create a child of the
>>> container's cgroup, that's probably great.
>>>
>> I thought of doing something like this in libcg (having a daemon and a
>> client socket interface), but dropped the idea later. When all
>> controllers support multi-levels well, the plan is to create a
>> sub-directory in the cgroup hierarchy and give subtree ownership to
>> the application administrator.
>>
>>> So I'm basically being quiet until I see whether libcg will suffice.
>>>
>> If you do have any specific requirements, we can cater to them right
>> now. Please do let us know. The biggest challenge right now is getting
>> a stable API.
>
> It sounds like what you're talking about should suffice - the container
> can only write to its own subdirectory, and the control files therein
> should not allow the container to escape the bounds set for it, only to
> partition it.

```

>
> The only thing that worries me is how subtle it may turn out to be to
> properly set up a container this way. I.e. you'll need to
> mount --bind /etc/cgroups/mycontainer /vps/container1/etc/cgroups
> before the container is off and running and be able to then prevent
> the cgroup from mounting the host's /etc any other way.
>
> As in so many other cases it shouldn't be too difficult with selinux,
> otherwise I suppose one thing you could do is to put the host's
> /etc/cgroup (or really the host's /) on partitionN, mount
> /etc/cgroup/container from another partitionM, and use the device
> whitelist (eventually, device namespaces) to allow the container to
> mount partitionM but not partitionN.
>
> So that's the one place where kernel support might be kind of seductive,
> but I suspect it would just lead to either an unsafe, an inflexible, or
> just a hokey "solution". So let's stick with libcg for now. A daemon
> can always be written on top of it if people want, and if at some point
> we see a real need for kernel support we can talk about it then.
>

Sounds fair to me. We intend to provide the basis for building a good daemon if ever required. You see left overs in libcg.h (that I need to clean up).

> Thanks, Balbir.

>
>>> > 5) "procs" control file
>>> > -----
>>> >
>>> > This would be the equivalent of the "tasks" file, but acting/reporting
>>> > on entire thread groups. Not sure exactly what the read semantics
>>> > should be if a sub-thread of a process is in the cgroup, but not its
>>> > thread group leader.
>>> >
>>> >
>>> > 6) Statistics / binary API
>>> > ----
>>> >
>>> > Balaji Rao is working on a generic way to gather per-subsystem
>>> > statistics; it would also be interesting to construct an extensible
>>> > binary API via taskstats. One possible way to do this (taken from my
>>> > email earlier today) would be:
>>> >
>>> > With the taskstats interface, we could have operations to:
>>> >
>>> > - describe the API exported by a given subsystem (automatically
>>> > generated, based on its registered control files and their access
>>> > methods)

```

>>> >
>>> > - retrieve a specified set of stats in a binary format
>>> >
>>> > So as a concrete example, with the memory, cpuacct and cpu subsystems
>>> > configured, the reported API might look something like (in pseudo-code
>>> > form)
>>> >
>>> > 0 : memory.usage_in_bytes : u64
>>> > 1 : memory.limit_in_bytes : u64
>>> > 2 : memory.failcnt : u64
>>> > 3 : memory.stat : map
>>> > 4 : cpuacct.usage : u64
>>> > 5 : cpu.shares : u64
>>> > 6 : cpu.rt_runtime_ms : s64
>>> > 7 : cpu.stat : map
>>> >
>>> > This list would be auto-generated by cgroups based on inspection of
>>> > the control files.
>>> >
>>> > The user could then request stats 0, 3 and 7 for a cgroup to get the
>>> > memory.usage_in_bytes, memory.stat and cpu.stat statistics.
>>> >
>>> > The stats could be returned in a binary format; the format for each
>>> > individual stat would depend on the type of that stat, and these could
>>> > be simply concatenated together.
>>> >
>>> > A u64 or s64 stat would simply be a 64-bit value in the data stream
>>> >
>>> > A map stat would be represented as a sequence of 64-bit values,
>>> > representing the values in the map. There would be no need to include
>>> > the size of the map or the key ordering in the binary format, since
>>> > userspace could determine that by reading the ASCII version of the map
>>> > control file once at startup.
>>> >
>>> > So in the case of the request above for stats 0, 3 & 7, the binary
>>> > stats stream would be a sequence of 64-bit values consisting of:
>>> >
>>> > <memory.usage>
>>> > <memory.stat.cache>
>>> > <memory.stat.rss>
>>> > <memory.stat.active>
>>> > <memory.stat.inactive>
>>> > <cpu.stat.utime>
>>> > <cpu.stat.stime>
>>> >
>>> > If more stats were added to memory.stat or cpu.stat by a future
>>> > version of the code, then they would automatically appear; any that
>>> > userspace didn't understand it could ignore.

```

```

>>> >
>>> > The userspace side of this could be handled by libcg.
>>> >
>> Yes, it can be easily handled by libcg. I think this is an important
>> piece of the cgroup infrastructure.
>>
>>> > 8) Subsystems from modules
>>> > -----
>>> >
>>> > Having completely unknown subsystems registered at run time would
>>> > involve adding a bunch of complexity and additional locking to cgroups
>>> > - but allowing a subsystem to be known at compile time but just
>>> > stubbed until first mounted (at which time its module would be loaded)
>>> > should increase the flexibility of cgroups without hurting its
>>> > complexity or performance.
>>> >
>>> >
>>> > 7) New subsystems
>>> > -----
>>> >
>>> > - Swap, disk I/O - already being worked on by others
>>> >
>>> > - OOM handler. Exactly what semantics this should provide aren't 100%
>>> > clear. At Google we have a useful OOM handler that allows root to
>>> > intercept OOMs as they're about to happen, and take appropriate action
>>> > such as killing some other lower-priority job to free up memory, etc.
>>> > Another useful feature of this subsystem might be to allow a process
>>> > in that cgroup to get an early notification that its cgroup is getting
>>> > close to OOM. This needs to be a separate subsystem since it could be
>>> > used to provide OOM notification/handling for localized OOMs caused
>>> > either by cpusets or the memory controller.
>>> >
>>> > - network tx/rx isolation. The cleanest way that we've found to do
>>> > this is to provide a per-cgroup id which can be exposed as a traffic
>>> > filter for regular Linux traffic control - then you can construct
>>> > arbitrary network queueing structures without requiring any new APIs,
>>> > and tie flows from particular cgroups into the appropriate queues.
>>> >
>>> >
>>> > 8) per-mm owner field
>>> > ----
>>> >
>>> > To remove the need for per-subsystem counted references from the mm.
>>> > Being developed by Balbir Singh
>>>
>> I have version 9 out. It has all the review comments incorporated. If
>> the patch seems reasonable, I'll ask Andrew to include it.
>>

```

>>> I'm slooowly trying to whip together a swapfile namespace - not a
>>> cgroup - which ties a swapfns to a list of swapfiles (where each
>>> swapfile belongs to only one swapfns). So I also need an mm->task
>>> pointer of some kind. I've got my own in my patches right now but
>>> sure do hope to make use of Balbir's mm owner field.
>> Serge, do you have any specific requirements for the mm owner field.
>> Will the current patch meet your requirements (including
>> mm_owner_changed field callbacks)?
>
> I'm behind in versions, but the last I took a look it looked great.

Thanks, that would be nice. I've just asked Andrew to include it, if there are no objections.

>
> thanks,
> -serge
>
> _____
> Containers mailing list
> Containers@lists.linux-foundation.org
> <https://lists.linux-foundation.org/mailman/listinfo/containers>

--

Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

Containers mailing list
Containers@lists.linux-foundation.org
<https://lists.linux-foundation.org/mailman/listinfo/containers>
