Subject: [RFC] Prefixing cgroup generic control filenames with "cgroup."
Posted by Paul Menage on Thu, 28 Feb 2008 21:14:05 GMT
View Forum Message <> Reply to Message

All control files created by cgroup subsystems are given a prefix
corresponding to their subsystem name. But control files provided by
cgroups itself have no prefix. Currently that set of files is just
"tasks", "notify_on_release" and "release_agent", but that set is
likely to expand in the future. To reduce the risk of clashes, it
would make sense to prefix these files and any future ones with the
"cgroup." prefix.

The only reason that I can see *not* to do this would be for
compatibility with 2.6.24. Do people think this is a strong enough
reason to leave the existing names? If distros are planning to ship
products based on 2.6.24, presumably they'd be adding their own
patches anyway, so they could add a trivial prefix change patch too.
(I realise this discussion would have been more useful *before* 2.6.24
shipped, but I didn't quite get round to it ...)

A compromise might be to keep "tasks" unprefixed, and say that future
names get the "cgroup." prefix; in this case I'd be inclined to add
the prefix to notify_on_release and release_agent on the grounds that
there's much less chance of breaking anyone with those files since (I
suspect) they're much less used.

Note that if you mount a cgroup filesystem with the "noprefix" option,
which is what the cpuset filesystem wrapper does, no subsystems have
prefixes, and in this case the "cgroup." prefix wouldn't be used
either. So this doesn't affect any users that explicitly mount cpusets
rather than cgroups.

Thoughts?

Paul

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

Subject: Re: [RFC] Prefixing cgroup generic control filenames with "cgroup."
Posted by akpm on Thu, 28 Feb 2008 21:21:42 GMT
View Forum Message <> Reply to Message

On Thu, 28 Feb 2008 13:14:05 -0800
"Paul Menage" <menage@google.com> wrote:

> All control files created by cgroup subsystems are given a prefix
> corresponding to their subsystem name. But control files provided by
> cgroups itself have no prefix. Currently that set of files is just
> "tasks", "notify_on_release" and "release_agent", but that set is
> likely to expand in the future. To reduce the risk of clashes, it
> would make sense to prefix these files and any future ones with the
> "cgroup." prefix.
>
> The only reason that I can see *not* to do this would be for
> compatibility with 2.6.24. Do people think this is a strong enough
> reason to leave the existing names? If distros are planning to ship
> products based on 2.6.24, presumably they'd be adding their own
> patches anyway, so they could add a trivial prefix change patch too.
> (I realise this discussion would have been more useful *before* 2.6.24
> shipped, but I didn't quite get round to it ...)
>
> A compromise might be to keep "tasks" unprefixed, and say that future
> names get the "cgroup." prefix; in this case I'd be inclined to add
> the prefix to notify_on_release and release_agent on the grounds that
> there's much less chance of breaking anyone with those files since (I
> suspect) they're much less used.
>
> Note that if you mount a cgroup filesystem with the "noprefix" option,
> which is what the cpuset filesystem wrapper does, no subsystems have
> prefixes, and in this case the "cgroup." prefix wouldn't be used
> either. So this doesn't affect any users that explicitly mount cpusets
> rather than cgroups.
>
> Thoughts?
>

It would be easier to judge if we could see the full directory tree.

Because if something is in /foo/bar/cgroup/notify_on_release then
prefixing the filename with "cgroup_" seems pretty pointless.

Subject: Re: [RFC] Prefixing cgroup generic control filenames with "cgroup."
Posted by Paul Menage on Thu, 28 Feb 2008 21:28:31 GMT
View Forum Message <> Reply to Message

On Thu, Feb 28, 2008 at 1:21 PM, Andrew Morton
<akpm@linux-foundation.org> wrote:

>
> Because if something is in /foo/bar/cgroup/notify_on_release then
> prefixing the filename with "cgroup_" seems pretty pointless.
>

The point would be to avoid situations where a user has code that
creates a group directory called "foo", and then in a future kernel
release cgroups introduces a control file called "foo". If it's
prefixed, then the user just has to avoid creating groups prefixed by
"cgroup." or any subsystem name, so collisions will be less likely.

Paul

Subject: Re: [RFC] Prefixing cgroup generic control filenames with "cgroup."
Posted by serge on Thu, 28 Feb 2008 21:28:35 GMT
View Forum Message <> Reply to Message

Quoting Paul Menage (menage@google.com):
> All control files created by cgroup subsystems are given a prefix
> corresponding to their subsystem name. But control files provided by
> cgroups itself have no prefix. Currently that set of files is just
> "tasks", "notify_on_release" and "release_agent", but that set is
> likely to expand in the future. To reduce the risk of clashes, it
> would make sense to prefix these files and any future ones with the
> "cgroup." prefix.
>
> The only reason that I can see *not* to do this would be for
> compatibility with 2.6.24. Do people think this is a strong enough
> reason to leave the existing names? If distros are planning to ship
> products based on 2.6.24, presumably they'd be adding their own
> patches anyway, so they could add a trivial prefix change patch too.
> (I realise this discussion would have been more useful *before* 2.6.24
> shipped, but I didn't quite get round to it ...)
>
> A compromise might be to keep "tasks" unprefixed, and say that future
> names get the "cgroup." prefix; in this case I'd be inclined to add
> the prefix to notify_on_release and release_agent on the grounds that
> there's much less chance of breaking anyone with those files since (I
> suspect) they're much less used.
>
> Note that if you mount a cgroup filesystem with the "noprefix" option,
> which is what the cpuset filesystem wrapper does, no subsystems have

> prefixes, and in this case the "cgroup." prefix wouldn't be used
> either. So this doesn't affect any users that explicitly mount cpusets
> rather than cgroups.
>
> Thoughts?

To me it seems quite logical that files belonging to the cgroup
subsystem would have no prefix, and I don't see any good reason to
do so.

-serge

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

---

Subject: Re: [RFC] Prefixing cgroup generic control filenames with "cgroup."
Posted by serge on Thu, 28 Feb 2008 21:33:19 GMT
View Forum Message <> Reply to Message

Quoting Paul Menage (menage@google.com):
> On Thu, Feb 28, 2008 at 1:21 PM, Andrew Morton
> <akpm@linux-foundation.org> wrote:
>
> >
> > Because if something is in /foo/bar/cgroup/notify_on_release then
> > prefixing the filename with "cgroup_" seems pretty pointless.
> >
>
> The point would be to avoid situations where a user has code that
> creates a group directory called "foo", and then in a future kernel
> release cgroups introduces a control file called "foo". If it's
> prefixed, then the user just has to avoid creating groups prefixed by
> "cgroup." or any subsystem name, so collisions will be less likely.

Have you already run into that case?

You said the set of files belong to cgroup itself is likely to increase
- do you have some candidates in mind? Perhaps ones which are likely
to conflict with choice taskgroup names?

If anything I'd say add a 'prefix_cgroup' mount option and use it to
decide whether to prefix or not (rather than use the noprefix option).

-serge

_____
Containers mailing list

Subject: Re: [RFC] Prefixing cgroup generic control filenames with "cgroup."
Posted by akpm on Thu, 28 Feb 2008 21:40:27 GMT
View Forum Message <> Reply to Message

On Thu, 28 Feb 2008 13:28:31 -0800
"Paul Menage" <menage@google.com> wrote:

> On Thu, Feb 28, 2008 at 1:21 PM, Andrew Morton
> <akpm@linux-foundation.org> wrote:
>
> > > Because if something is in /foo/bar/cgroup/notify_on_release then
> > > prefixing the filename with "cgroup_" seems pretty pointless.
> > >
>
> The point would be to avoid situations where a user has code that
> creates a group directory called "foo", and then in a future kernel
> release cgroups introduces a control file called "foo". If it's
> prefixed, then the user just has to avoid creating groups prefixed by
> "cgroup." or any subsystem name, so collisions will be less likely.
>

Maybe cgroups shouldn't be putting kernel-generated files in places where
user-specified files appear?

(Am still thrashing around a bit here without an overview of the overall
layout and naming).

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

Subject: Re: [RFC] Prefixing cgroup generic control filenames with "cgroup."
Posted by Paul Menage on Thu, 28 Feb 2008 22:06:30 GMT
View Forum Message <> Reply to Message

On Thu, Feb 28, 2008 at 1:40 PM, Andrew Morton
<akpm@linux-foundation.org> wrote:
>
> Maybe cgroups shouldn't be putting kernel-generated files in places where
> user-specified files appear?

>

Well, that API (mixing control files and group directories in the same directory namespace) was inherited directly from cpusets.

It wouldn't be hard to throw that away and move all the user-created group directories into their own subdirectory, i.e. change the existing directory layout from something like:

```
/mnt/cgroup/
   tasks
   cpu.shares
   memory.limit_in_bytes
   memory.usage_in_bytes
   user_created_groupname1/
      tasks
      cpu.shares
      memory.limit_in_bytes
      memory.usage_in_bytes
   user_created_groupname2/
      tasks
      cpu.shares
      memory.limit_in_bytes
      memory.usage_in_bytes
```

to something like:

```
/mnt/cgroup/
   tasks
   cpu.shares
   memory.limit_in_bytes
   memory.usage_in_bytes
   groups/
      user_created_groupname1/
         tasks
         cpu.shares
         memory.limit_in_bytes
         memory.usage_in_bytes
         groups/
      user_created_groupname2/
         tasks
         cpu.shares
         memory.limit_in_bytes
         memory.usage_in_bytes
         groups/
```

That would completely solve the namespace problem, at the cost of a little extra verbosity/inelegance for human users (I suspect

programmatic users would prefer it), and lack of compatibility with
2.6.24. I'd also need to make the existing model a mount option so
that we could keep compatibility with the cpusets filesystem API.

> (Am still thrashing around a bit here without an overview of the overall
> layout and naming).

Pretty much the same as cpusets, other than the additional
kernel-generated files in each directory, as provided by the resource
subsystems. So the same potential problem faced cpusets, but the fact
that new cpuset features weren't being developed quickly meant the
problem was less likely to actually bite people.

Paul

Subject: Re: [RFC] Prefixing cgroup generic control filenames with "cgroup."
Posted by Paul Menage on Thu, 28 Feb 2008 22:06:44 GMT
View Forum Message <> Reply to Message

On Thu, Feb 28, 2008 at 1:33 PM, <serge@hallyn.com> wrote:
>
> You said the set of files belong to cgroup itself is likely to increase
> - do you have some candidates in mind?

Nothing concrete right now. One example that I already proposed was
the "cgroup.api" file but that's shelved for now, until such time as I
actually propose the binary API that it was intended to help support.

> Perhaps ones which are likely
> to conflict with choice taskgroup names?

It's hard to determine what likely taskgroup names would be. For my
own use, pretty much every group has a unique 64-bit ID in the name so
this isn't something I worry about directly affecting our systems. But
I don't know what other users might want to do.

>
> If anything I'd say add a 'prefix_cgroup' mount option and use it to
> decide whether to prefix or not (rather than use the noprefix option).

The existing "noprefix" option controls whether the *subsystems* get
prefixes. It's mainly there to provide backwards compatibility for
cpusets. Existing cpusets clients would be expecting to find files

with names like "mems_allowed" rather than "cpuset.mems_allowed". So
mounting with the "noprefix" option (which happens automatically when
you mount the "cpuset" filesystem wrapper) gives the same result as
before.

Currently "noprefix" has no effect on cgroup files, since they never
have a prefix anyway.

Yes, we could add a new mount option "prefixcgroup", and let people
decide which they want. But I still don't see any argument *against*
doing the prefixing automatically (rather than an argument that it's
no better or worse) other than wanting 2.6.24 compatibility.

Paul

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

---

Subject: Re: [RFC] Prefixing cgroup generic control filenames with "cgroup."
Posted by akpm on Thu, 28 Feb 2008 22:21:00 GMT
View Forum Message <> Reply to Message

On Thu, 28 Feb 2008 14:06:30 -0800
"Paul Menage" <menage@google.com> wrote:

> On Thu, Feb 28, 2008 at 1:40 PM, Andrew Morton
> <akpm@linux-foundation.org> wrote:
> >
> > Maybe cgroups shouldn't be putting kernel-generated files in places where
> > user-specified files appear?
> >
>
> Well, that API (mixing control files and group directories in the same
> directory namespace) was inherited directly from cpusets.
>
> It wouldn't be hard to throw that away and move all the user-created
> group directories into their own subdirectory, i.e. change the
> existing directory layout from something like:
>
> /mnt/cgroup/
>   tasks
>   cpu.shares
>   memory.limit_in_bytes
>   memory.usage_in_bytes
>   user_created_groupname1/
>   tasks

>   cpu.shares
>   memory.limit_in_bytes
>   memory.usage_in_bytes
>   user_created_groupname2/
>   tasks
>   cpu.shares
>   memory.limit_in_bytes
>   memory.usage_in_bytes
>
> to something like:
>
> /mnt/cgroup/
>   tasks
>   cpu.shares
>   memory.limit_in_bytes
>   memory.usage_in_bytes
>   groups/
>   user_created_groupname1/
>   tasks
>   cpu.shares
>   memory.limit_in_bytes
>   memory.usage_in_bytes
>   groups/
>   user_created_groupname2/
>   tasks
>   cpu.shares
>   memory.limit_in_bytes
>   memory.usage_in_bytes
>   groups/

That looks nice.

> That would completely solve the namespace problem, at the cost of a
> little extra verbosity/inelegance for human users (I suspect
> programmatic users would prefer it), and lack of compatibility with
> 2.6.24. I'd also need to make the existing model a mount option so
> that we could keep compatibility with the cpusets filesystem API.

That doesn't. It sounds like cpusets legacy has mucked us up here?

Could we do something like auto-prefixing user-created directories with a
fixed string so that there is no way in which the user can cause a
collision with kernel-created files?

I suppose that would break cpusets back-compatibility as well? If so, we
could do the prefixing only for non-cpusets directories, but that's getting
a bit weird.

> > (Am still thrashing around a bit here without an overview of the overall
> > layout and naming).
>
> Pretty much the same as cpusets, other than the additional
> kernel-generated files in each directory, as provided by the resource
> subsystems. So the same potential problem faced cpusets, but the fact
> that new cpuset features weren't being developed quickly meant the
> problem was less likely to actually bite people.

hm. I guess that all the kernel-generated file names are known up-front
and that they are instantiated early, so if a user tried to create a cgroup
called "tasks", than that would just fail.

But, as you say, later addition of new kernel-created files might collide
with prior userspace installations.

So yet another option would be to promise to prefix all _future_
kernel-generated files with "kern_", and to change the implementation now
to reject any user-created files which start with "kern_". hm.

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

---

Subject: Re: [RFC] Prefixing cgroup generic control filenames with "cgroup."
Posted by Paul Menage on Thu, 28 Feb 2008 22:26:39 GMT
View Forum Message <> Reply to Message

On Thu, Feb 28, 2008 at 2:21 PM, Andrew Morton
<akpm@linux-foundation.org> wrote:
> On Thu, 28 Feb 2008 14:06:30 -0800
>
> "Paul Menage" <menage@google.com> wrote:
>
>
>
> > On Thu, Feb 28, 2008 at 1:40 PM, Andrew Morton
> > <akpm@linux-foundation.org> wrote:
> > >
> > > Maybe cgroups shouldn't be putting kernel-generated files in places where
> > > user-specified files appear?
> > >
> >
> > Well, that API (mixing control files and group directories in the same
> > directory namespace) was inherited directly from cpusets.
> >

> > It wouldn't be hard to throw that away and move all the user-created
> > group directories into their own subdirectory, i.e. change the
> > existing directory layout from something like:
> >
> > /mnt/cgroup/
> > tasks
> > cpu.shares
> > memory.limit_in_bytes
> > memory.usage_in_bytes
> > user_created_groupname1/
> > tasks
> > cpu.shares
> > memory.limit_in_bytes
> > memory.usage_in_bytes
> > user_created_groupname2/
> > tasks
> > cpu.shares
> > memory.limit_in_bytes
> > memory.usage_in_bytes
> >
> > to something like:
> >
> > /mnt/cgroup/
> > tasks
> > cpu.shares
> > memory.limit_in_bytes
> > memory.usage_in_bytes
> > groups/
> > user_created_groupname1/
> > tasks
> > cpu.shares
> > memory.limit_in_bytes
> > memory.usage_in_bytes
> > groups/
> > user_created_groupname2/
> > tasks
> > cpu.shares
> > memory.limit_in_bytes
> > memory.usage_in_bytes
> > groups/
>
> That looks nice.
>

I'll put a patch together for consideration.

> Could we do something like auto-prefixing user-created directories with a
> fixed string so that there is no way in which the user can cause a

> collision with kernel-created files?

That's something like putting them all in their own sub-directory, but
sounds less clean to me.

>
> I suppose that would break cpusets back-compatibility as well? If so, we
> could do the prefixing only for non-cpusets directories, but that's getting
> a bit weird.

We already have something like that in place, actually.

When you mount the legacy "cpuset" filesystem, it just passes through
to the cgroup filesystem with the mount options "cpuset,noprefix",
i.e. mount a cgroups hierarcy with just cpusets bound to it, and
don't prefix subsystem control files with the subsystem name. It
wouldn't be hard to also have a "nosubdir" mount option that keeps the
existing single-level style, have the cpuset filessytem pass that
option.

Paul

Subject: Re: [RFC] Prefixing cgroup generic control filenames with "cgroup."
Posted by Paul Jackson on Thu, 28 Feb 2008 23:36:18 GMT
View Forum Message <> Reply to Message

Paul M wrote:
> But control files provided by
> cgroups itself have no prefix. Currently that set of files is just
> "tasks", "notify_on_release" and "release_agent", but that set is
> likely to expand in the future. To reduce the risk of clashes, it
> would make sense to prefix these files and any future ones with the
> "cgroup." prefix.

I don't see the mixing of kernel generated filenames with user
generated names to be a practical problem. There just aren't that many
names in play here.

I'd think that renaming even the few cgroup files that were published
in 2.6.24 would be a fairly unacceptable incompatibility.

Paul M wrote:
> The point would be to avoid situations where a user has code that

> creates a group directory called "foo", and then in a future kernel
> release cgroups introduces a control file called "foo".

We could accomplish that much by decreeing that future new kernel
generated names that we might add follow some stronger convention,
such as the cgroup_ or appropriate subsystem prefix. No need to
change the existing well known names for this reason.

Serge wrote:
> To me it seems quite logical that files belonging to the cgroup
> subsystem would have no prefix, and I don't see any good reason to
> do so.

Agreed.

Paul M wrote:
> It wouldn't be hard to throw that away and move all the user-created
> group directories into their own subdirectory, i.e. change the
> existing directory layout from something like:

Yuck. You're complicating this more than necessary, to solve a problem
that exists only in your imagination ;). Simple, overlapping,
namespaces really are ok, so long as the number of distinct names and
the rate of their growth are sufficiently low.

> But I don't know what other users might want to do.

Just set some convention for future added names; that's enough to enable
others adding user space names to avoid collisions. Such as using only
lower case letters and underscores, or the cgroup_ or other prefix
on -future- names. That's sufficient, if not more than sufficient.

> But I still don't see any argument *against*
> doing the prefixing automatically (rather than an argument that it's
> no better or worse) other than wanting 2.6.24 compatibility.

That's sufficient reason. Actually, in terms of 'common names used
by humans' some of these names, "tasks" and "notify_on_release", date
back much earlier than that. Please don't rename these two files in
cgroups; and of course absolutely don't rename them in cpusets.

Please don't end up with different names of these files, depending on
whether you're in cgroups or cpusets, either.

Andrew, looking at a tree that Paul M drew, wrote:
> That looks nice.

Not to me ;) Yuck.

Andrew wrote:
> That doesn't. It sounds like cpusets legacy has mucked us up here?
>
> Could we do something like auto-prefixing user-created directories with a
> fixed string so that there is no way in which the user can cause a
> collision with kernel-created files?

Lordy lordy -- a bunch of intrusive, complicating crap to solve a
non-existent problem (sorry for the indelicate choice of words ;).

> So yet another option would be to promise to prefix all _future_
> kernel-generated files with "kern_", ...

that could work

> ... and to change the implementation now
> to reject any user-created files which start with "kern_". hm.

Unnecessary complication. No need to burden our children and grand children
with this special case, forever after, in order to solve some empty set of
special cases currently facing us.

> > It wouldn't be hard to throw that away and move all the user-created
> > group directories into their own subdirectory, i.e. change the
> > existing directory layout from something like:
> ...
> I'll put a patch together for consideration.

I'll keep a bucket of ice water handy, for that patch ;). One nice
property of the cpuset file system is that there is exactly one
directory per cpuset. The kernel creates regular files; the user
creates directories; no exceptions. I encourage us to keep that state
of affairs, for both cgroups and cpusets.

> > If so, we could do the prefixing only for non-cpusets directories,
> > but that's getting a bit weird.
>
> We already have something like that in place, actually.

I probably won't insist on a full force NAK so long as cpusets are
unchanged, thanks to such compatibility measures; though I would be
tempted to ... if I could think of a sufficient reason.

Human beings really don't have problems with overlapping name spaces
like this; to them it seems simpler in many cases, such as this one.

This has been, in my (biased and less than humble) view, one of the

successes of the cpuset interface. It's just enough to comfortably do what's needed; not some overly formalized and unnecessarily robust complication beyond what's practically needed. More people can comfortably understand it this way.

Design interfaces, and write code, for humans. Resists making concessions to the limitations of computers (or our fellow geeks) as best you can, whenever you can.

--

I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.940.382.4214

_____

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

---

Subject: Re: [RFC] Prefixing cgroup generic control filenames with "cgroup."
Posted by Paul Menage on Fri, 29 Feb 2008 01:03:33 GMT
View Forum Message <> Reply to Message

On Thu, Feb 28, 2008 at 3:36 PM, Paul Jackson <pj@sgi.com> wrote:
>
> We could accomplish that much by decreeing that future new kernel
> generated names that we might add follow some stronger convention,
> such as the cgroup_ or appropriate subsystem prefix.

Subsystem-created files already have an appropriate prefix.

> No need to
> change the existing well known names for this reason.

But that's part of my point - is it reasonable to describe a system that was only introduced in 2.6.24 as "well-known"?

>
> Actually, in terms of 'common names used
> by humans' some of these names, "tasks" and "notify_on_release", date
> back much earlier than that. Please don't rename these two files in
> cgroups; and of course absolutely don't rename them in cpusets.

No, I wasn't planning to make any changes to cpusets.

>
> Please don't end up with different names of these files, depending on
> whether you're in cgroups or cpusets, either.

That already happens - when mounted as the "cpuset" filesystem, we
have names like "mems_allowed". When mounted as cgroups, we have names
like cpuset.mems_allowed.

>
> > Could we do something like auto-prefixing user-created directories with a
> > fixed string so that there is no way in which the user can cause a
> > collision with kernel-created files?
>
> Lordy lordy -- a bunch of intrusive, complicating crap to solve a
> non-existent problem (sorry for the indelicate choice of words ;).

No, I don't like that idea either.

Paul

Subject: Re: [RFC] Prefixing cgroup generic control filenames with "cgroup."
Posted by Paul Jackson on Fri, 29 Feb 2008 01:22:32 GMT
View Forum Message <> Reply to Message

> Subsystem-created files already have an appropriate prefix.

Good ... such little consistencies in naming are helpful, when
voluntarily self imposed, without incompatible changes to published
names.

> > No need to
> > change the existing well known names for this reason.
>
> But that's part of my point - is it reasonable to describe a system
> that was only introduced in 2.6.24 as "well-known"?

In their earlier cpuset context, "tasks" and "notify_on_release" have
been well known for years.

And breaking actual compatibility, in a way that will break more than
the tiniest set of users, even over one release, should not be done
without both (1) a really good cause, and (2) a plan for migrating
users over a couple of releases to the new interface.

> > back much earlier than that. Please don't rename these two files in
> > cgroups; and of course absolutely don't rename them in cpusets.

>
> No, I wasn't planning to make any changes to cpusets.

Yeah - I figured you wouldn't risk my wrath changing this in cpusets ;).

Could you please not change them in cgroups, either?

> > Please don't end up with different names of these files, depending on
> > whether you're in cgroups or cpusets, either.
>
> That already happens - when mounted as the "cpuset" filesystem, we
> have names like "mems_allowed". When mounted as cgroups, we have names
> like cpuset.mems_allowed.

Ok - it does make sense that cpuset specific files, when embedded in the
more general purpose cgroups, are adorned with name prefixes that they
didn't have in the legacy API. And besides, it is what is -- we released
it, and there's no compelling disaster forcing a change.

But generic cgroup infrastructure files, such as "tasks", have not had
such adornments. And now, that is what it is -- released, and sufficient
to remain as it is.

> No, I don't like that idea either.

Good (and extra credit for saying so with fewer words and more politely
than I managed ;).

--
I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.940.382.4214

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

---

Subject: [RFC] [PATCH] Re: Prefixing cgroup generic control filenames with "cgroup."
Posted by Paul Menage on Fri, 29 Feb 2008 05:59:21 GMT
View Forum Message <> Reply to Message

>> to something like:
>> >
>> > /mnt/cgroup/
>> >   tasks
>> >   cpu.shares

>> >   memory.limit_in_bytes
>> >   memory.usage_in_bytes
>> >   groups/
>> >   user_created_groupname1/
>> >   tasks
>> >   cpu.shares
>> >   memory.limit_in_bytes
>> >   memory.usage_in_bytes
>> >   groups/
>> >   user_created_groupname2/
>> >   tasks
>> >   cpu.shares
>> >   memory.limit_in_bytes
>> >   memory.usage_in_bytes
>> >   groups/
>
> That looks nice.
>

OK, here's a patch that does that. It's not quite right yet, as it crashes on unmount, but the basic idea is there. It adds the additional "groups" directory by default, but uses the previous behaviour if the nogroupdir option is passed (done by default for cpusets).

Thoughts? Is this a direction we want to go in? As an option, or by default?

Paul

```
 include/linux/cgroup.h |    2
 kernel/cgroup.c        |  164 ++++++++++++++++++++++++++++++++++++-------------------
 kernel/cpuset.c        |    2
 3 files changed, 122 insertions(+), 46 deletions(-)

Index: subdir-2.6.25-rc3/include/linux/cgroup.h
===================================================================
--- subdir-2.6.25-rc3.orig/include/linux/cgroup.h
+++ subdir-2.6.25-rc3/include/linux/cgroup.h
@@ -105,7 +105,7 @@ struct cgroup {

  struct cgroup *parent; /* my parent */
  struct dentry *dentry;    /* cgroup fs entry */
-
+ struct dentry *group_dentry;
  /* Private pointers for each registered subsystem */
  struct cgroup_subsys_state *subsys[CGROUP_SUBSYS_COUNT];

Index: subdir-2.6.25-rc3/kernel/cgroup.c
===================================================================
```

```
--- subdir-2.6.25-rc3.orig/kernel/cgroup.c
+++ subdir-2.6.25-rc3/kernel/cgroup.c
@@ -139,6 +139,7 @@ inline int cgroup_is_removed(const struc
 /* bits in struct cgroupfs_root flags field */
 enum {
  ROOT_NOPREFIX, /* mounted subsystems have no named prefix */
+ ROOT_NOGROUPDIR, /* user-created sub-groups go in the main directory */
 };

 static int cgroup_is_releasable(const struct cgroup *cgrp)
@@ -475,6 +476,16 @@ static struct css_set *find_css_set(
  return res;
 }

+static inline struct cgroup *__d_cgrp(struct dentry *dentry)
+{
+ return dentry->d_fsdata;
+}
+
+static inline struct cftype *__d_cft(struct dentry *dentry)
+{
+ return dentry->d_fsdata;
+}
+
 /*
  * There is one global cgroup mutex. We also require taking
  * task_lock() when dereferencing a task's cgroup subsys pointers.
@@ -598,33 +609,41 @@ static void cgroup_diput(struct dentry *
  /* is dentry a directory ? if so, kfree() associated cgroup */
  if (S_ISDIR(inode->i_mode)) {
   struct cgroup *cgrp = dentry->d_fsdata;
-  struct cgroup_subsys *ss;
-  BUG_ON(!(cgroup_is_removed(cgrp)));
-  /* It's possible for external users to be holding css
-   * reference counts on a cgroup; css_put() needs to
-   * be able to access the cgroup after decrementing
-   * the reference count in order to know if it needs to
-   * queue the cgroup to be handled by the release
-   * agent */
-  synchronize_rcu();
+  if (dentry != cgrp->group_dentry) {
+   struct cgroup_subsys *ss;
+   BUG_ON(!(cgroup_is_removed(cgrp)));
+   /*
+    * It's possible for external users to be
+    * holding css reference counts on a cgroup;
+    * css_put() needs to be able to access the
+    * cgroup after decrementing the reference
```

```
+    * count in order to know if it needs to queue
+    * the cgroup to be handled by the release
+    * agent
+    */
+   synchronize_rcu();

-  mutex_lock(&cgroup_mutex);
-  /*
-   * Release the subsystem state objects.
-   */
-  for_each_subsys(cgrp->root, ss) {
-   if (cgrp->subsys[ss->subsys_id])
-    ss->destroy(ss, cgrp);
-  }
+   mutex_lock(&cgroup_mutex);
+   /*
+    * Release the subsystem state objects.
+    */
+   for_each_subsys(cgrp->root, ss) {
+    if (cgrp->subsys[ss->subsys_id])
+     ss->destroy(ss, cgrp);
+   }

-  cgrp->root->number_of_cgroups--;
-  mutex_unlock(&cgroup_mutex);
+   cgrp->root->number_of_cgroups--;
+   mutex_unlock(&cgroup_mutex);

-  /* Drop the active superblock reference that we took when we
-   * created the cgroup */
-  deactivate_super(cgrp->root->sb);
+   /*
+    * Drop the active superblock reference that
+    * we took when we created the cgroup
+    */
+   deactivate_super(cgrp->root->sb);

-  kfree(cgrp);
+   kfree(cgrp);
+  } else
+   cgrp->group_dentry = NULL;
   }
   iput(inode);
  }
@@ -638,24 +657,40 @@ static void remove_dir(struct dentry *d)
   dput(parent);
  }
```

```
-static void cgroup_clear_directory(struct dentry *dentry)
+static void cgroup_d_remove_dir(struct dentry *dentry);
+
+static void cgroup_clear_directory(struct dentry *dentry, int zap_groupdir)
 {
  struct list_head *node;
-
+ struct cgroup *cgrp;
  BUG_ON(!mutex_is_locked(&dentry->d_inode->i_mutex));
  spin_lock(&dcache_lock);
+ cgrp = __d_cgrp(dentry);
  node = dentry->d_subdirs.next;
  while (node != &dentry->d_subdirs) {
   struct dentry *d = list_entry(node, struct dentry, d_u.d_child);
+  if ((d == cgrp->group_dentry) && !zap_groupdir) {
+   node = node->next;
+   continue;
+  }
+
   list_del_init(node);
   if (d->d_inode) {
-   /* This should never be called on a cgroup
-    * directory with child cgroups */
-   BUG_ON(d->d_inode->i_mode & S_IFDIR);
    d = dget_locked(d);
    spin_unlock(&dcache_lock);
    d_delete(d);
-   simple_unlink(dentry->d_inode, d);
+   if (d == cgrp->group_dentry) {
+    mutex_lock(&d->d_inode->i_mutex);
+    cgroup_d_remove_dir(d);
+    mutex_unlock(&d->d_inode->i_mutex);
+    dput(dentry);
+    cgrp->group_dentry = NULL;
+   } else {
+    /* This should never be called on a cgroup
+     * directory with child cgroups */
+    BUG_ON(d->d_inode->i_mode & S_IFDIR);
+    simple_unlink(dentry->d_inode, d);
+   }
    dput(d);
    spin_lock(&dcache_lock);
   }
@@ -669,12 +704,13 @@ static void cgroup_clear_directory(struc
  */
 static void cgroup_d_remove_dir(struct dentry *dentry)
 {
- cgroup_clear_directory(dentry);
```

```
+ cgroup_clear_directory(dentry, 1);

  spin_lock(&dcache_lock);
  list_del_init(&dentry->d_u.d_child);
  spin_unlock(&dcache_lock);
  remove_dir(dentry);
+ dput(dentry);
 }

 static int rebind_subsystems(struct cgroupfs_root *root,
@@ -755,6 +791,8 @@ static int cgroup_show_options(struct se
   seq_printf(seq, ",%s", ss->name);
  if (test_bit(ROOT_NOPREFIX, &root->flags))
   seq_puts(seq, ",noprefix");
+ if (test_bit(ROOT_NOGROUPDIR, &root->flags))
+  seq_puts(seq, ",nogroupdir");
  if (strlen(root->release_agent_path))
   seq_printf(seq, ",release_agent=%s", root->release_agent_path);
  mutex_unlock(&cgroup_mutex);
@@ -785,6 +823,8 @@ static int parse_cgroupfs_options(char *
    opts->subsys_bits = (1 << CGROUP_SUBSYS_COUNT) - 1;
   } else if (!strcmp(token, "noprefix")) {
    set_bit(ROOT_NOPREFIX, &opts->flags);
+  } else if (!strcmp(token, "nogroupdir")) {
+   set_bit(ROOT_NOGROUPDIR, &opts->flags);
   } else if (!strncmp(token, "release_agent=", 14)) {
    /* Specifying two release agents is forbidden */
    if (opts->release_agent)
@@ -932,6 +972,8 @@ static int cgroup_get_rootdir(struct sup
  return 0;
 }

+static int cgroup_create_groupdir(struct cgroup *cgrp, int mode);
+
 static int cgroup_get_sb(struct file_system_type *fs_type,
     int flags, const char *unused_dev_name,
     void *data, struct vfsmount *mnt)
@@ -1050,6 +1092,8 @@ static int cgroup_get_sb(struct file_sys
   BUG_ON(!list_empty(&cgrp->children));
   BUG_ON(root->number_of_cgroups != 1);

+  cgroup_create_groupdir(cgrp, 0755);
+
   cgroup_populate_dir(cgrp);
   mutex_unlock(&inode->i_mutex);
   mutex_unlock(&cgroup_mutex);
@@ -1103,6 +1147,10 @@ static void cgroup_kill_sb(struct super_
  }
```

```
  mutex_unlock(&cgroup_mutex);

+ if (cgrp->group_dentry) {
+  dput(cgrp->group_dentry);
+  cgrp->group_dentry = NULL;
+ }
  kfree(root);
  kill_litter_super(sb);
 }
@@ -1113,16 +1161,6 @@ static struct file_system_type cgroup_fs
  .kill_sb = cgroup_kill_sb,
 };

-static inline struct cgroup *__d_cgrp(struct dentry *dentry)
-{
- return dentry->d_fsdata;
-}
-
-static inline struct cftype *__d_cft(struct dentry *dentry)
-{
- return dentry->d_fsdata;
-}
-
 /**
  * cgroup_path - generate the path of a cgroup
  * @cgrp: the cgroup in question
@@ -1538,13 +1576,17 @@ static struct file_operations cgroup_fil
  .release = cgroup_file_release,
 };

-static struct inode_operations cgroup_dir_inode_operations = {
+static struct inode_operations cgroup_groupdir_inode_operations = {
  .lookup = simple_lookup,
  .mkdir = cgroup_mkdir,
  .rmdir = cgroup_rmdir,
  .rename = cgroup_rename,
 };

+static struct inode_operations cgroup_dir_inode_operations = {
+ .lookup = simple_lookup,
+};
+
 static int cgroup_create_file(struct dentry *dentry, int mode,
     struct super_block *sb)
 {
@@ -1609,6 +1651,39 @@ static int cgroup_create_dir(struct cgro
  return error;
 }
```

```
+static int cgroup_create_groupdir(struct cgroup *cgrp, int mode)
+{
+ struct dentry *parent, *dentry;
+ int error = 0;
+
+ parent = cgrp->dentry;
+
+ if (test_bit(ROOT_NOGROUPDIR, &cgrp->root->flags)) {
+  parent->d_inode->i_op = &cgroup_groupdir_inode_operations;
+  return 0;
+ }
+
+ dentry = lookup_one_len("groups", parent, strlen("groups"));
+ if (!IS_ERR(dentry)) {
+  error = cgroup_create_file(dentry,
+         S_IFDIR | mode, cgrp->root->sb);
+  if (!error) {
+   dentry->d_fsdata = cgrp;
+   dentry->d_inode->i_op =
+    &cgroup_groupdir_inode_operations;
+   inc_nlink(parent->d_inode);
+   cgrp->group_dentry = dentry;
+   dget(dentry);
+   mutex_unlock(&dentry->d_inode->i_mutex);
+  }
+  dput(dentry);
+ } else {
+  error = PTR_ERR(dentry);
+ }
+
+ return error;
+}
+
 int cgroup_add_file(struct cgroup *cgrp,
          struct cgroup_subsys *subsys,
          const struct cftype *cft)
@@ -2169,8 +2244,8 @@ static int cgroup_populate_dir(struct cg
  int err;
  struct cgroup_subsys *ss;

- /* First clear out any existing files */
- cgroup_clear_directory(cgrp->dentry);
+ /* First clear out any existing control files */
+ cgroup_clear_directory(cgrp->dentry, 0);

  err = cgroup_add_files(cgrp, NULL, files, ARRAY_SIZE(files));
  if (err < 0)
```

```
@@ -2258,9 +2333,11 @@ static long cgroup_create(struct cgroup
  if (err < 0)
   goto err_remove;

+
  /* The cgroup directory was pre-locked for us */
  BUG_ON(!mutex_is_locked(&cgrp->dentry->d_inode->i_mutex));

+ cgroup_create_groupdir(cgrp, mode);
  err = cgroup_populate_dir(cgrp);
  /* If err < 0, we have a half-filled directory - oh well ;) */

@@ -2377,7 +2454,6 @@ static int cgroup_rmdir(struct inode *un
  spin_unlock(&d->d_lock);

  cgroup_d_remove_dir(d);
- dput(d);

  set_bit(CGRP_RELEASABLE, &parent->flags);
  check_for_release(parent);
Index: subdir-2.6.25-rc3/kernel/cpuset.c
===================================================================
--- subdir-2.6.25-rc3.orig/kernel/cpuset.c
+++ subdir-2.6.25-rc3/kernel/cpuset.c
@@ -243,7 +243,7 @@ static int cpuset_get_sb(struct file_sys
  int ret = -ENODEV;
  if (cgroup_fs) {
   char mountopts[] =
-   "cpuset,noprefix,"
+   "cpuset,noprefix,nogroupdir"
    "release_agent=/sbin/cpuset_release_agent";
   ret = cgroup_fs->get_sb(cgroup_fs, flags,
        unused_dev_name, mountopts, mnt);
```

Subject: Re: [RFC] [PATCH] Re: Prefixing cgroup generic control filenames with "cgroup."

Posted by Paul Jackson on Fri, 29 Feb 2008 06:20:51 GMT

View Forum Message <> Reply to Message

Paul M wrote:
> Thoughts? Is this a direction we want to go in? As an option, or by default?

So ... this proposal adds a 'groups' subdirectory in each cgroup, and places

the user generated subgroups in there.

It looks like an unnecessary, incompatible and complicating change to me.

For example, what would have been cgroup:

/mnt/cgroup/user_created_groupname1/user_created_groupname2

now becomes:

/mnt/cgroup/cgroups/user_created_groupname1/cgroups/user_created_groupname2

Right?

Why would you do this?

There is no problem with the current implementation, no bug we're having trouble coding a fix for, no feature we're have trouble adding. The current code, that simply doesn't allow colliding user names because the kernel provided names are already there, works just fine.

You're doing this just to "protect the user from themself", to make it more difficult for them to rely on some name that in a future version is no longer available.

It annoys users, and rightfully so, to have to permanently deal with interface warts, because the computer is trying to protect the user from some hypothetical scenario that is not a problem the user needs solving.

There is really a trivial resolution to this ... stake out what additional kernel generated names might ever be added ... some pattern(s) of characters which all future names will match, which leave wide swaths of names safely available, in perpetuity, for user created names, with no risk of future collision.

And did I say incompatible with released versions?

Hopefully Paul M isn't too surprised that I'm not endorsing this one ;).

--

I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.940.382.4214

Subject: Re: [RFC] [PATCH] Re: Prefixing cgroup generic control filenames with "cgroup."
Posted by Paul Menage on Fri, 29 Feb 2008 09:34:37 GMT
View Forum Message <> Reply to Message

On Thu, Feb 28, 2008 at 10:20 PM, Paul Jackson <pj@sgi.com> wrote:
>
> For example, what would have been cgroup:
>
> /mnt/cgroup/user_created_groupname1/user_created_groupname2
>
> now becomes:
>
> /mnt/cgroup/cgroups/user_created_groupname1/cgroups/user_created_groupname2
>
> Right?

Well, the additional components are called "groups" not "cgroups", but basically yes.

>
> You're doing this just to "protect the user from themself", to make
> it more difficult for them to rely on some name that in a future
> version is no longer available.

Correct. "Future-proofing" and "Forward planning" are two alternatively-nuanced ways of describing this ...

>
> There is really a trivial resolution to this ... stake out what
> additional kernel generated names might ever be added ... some
> pattern(s) of characters which all future names will match, which
> leave wide swaths of names safely available, in perpetuity, for
> user created names, with no risk of future collision.

Yes, we could just say "the kernel reserves the right to use any names that begin with a lower-case letter, and no others", and be done with it. That leaves a bit of an ugly taste in my mouth, but if people seem to prefer that approach we can go for it.

>
> And did I say incompatible with released versions?

Not at all incompatible if it requires a mount option to enable it ...

Paul

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [RFC] Prefixing cgroup generic control filenames with "cgroup."
Posted by xpl on Fri, 29 Feb 2008 11:38:20 GMT
View Forum Message <> Reply to Message

Paul Menage ??????:
> ...
>
> A compromise might be to keep "tasks" unprefixed, and say that future
> names get the "cgroup." prefix; in this case I'd be inclined to add
> the prefix to notify_on_release and release_agent on the grounds that
> there's much less chance of breaking anyone with those files since (I
> suspect) they're much less used.
>
This makes most sense to me. It won't break any existing software (most likely) while it seems reasonable to leave 'tasks' unprefixed as this is something that any software using any subsystem of cgroup would be using anyway and it is not that much associated with a particulat subsystem.

Regards,
Peter Litov.

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

Subject: Re: [RFC] [PATCH] Re: Prefixing cgroup generic control filenames with "cgroup."
Posted by Paul Jackson on Fri, 29 Feb 2008 15:30:13 GMT
View Forum Message <> Reply to Message

Paul M wrote:
> Well, the additional components are called "groups" not "cgroups", but
> basically yes.

Ah yes - "groups" - right, sorry:

/mnt/cgroup/groups/user_created_groupname1/groups/user_created_groupname2

> Yes, we could just say "the kernel reserves the right to use any names
> that begin with a lower-case letter, and no others", and be done with
> it.

The pattern might be stronger (more restrictive) than "[a-z].*"

The pattern might be something like:
 1) the known set of existing names (a short, specific list), plus
 2) "[a-z]+\.[a-z]+(_[a-z]+)*"

That second pattern is some lower case letters, a dot, and some
more lower case letters, possibly with some embedded underscores.

That is, except for the grandfathered existing known names, such as
"tasks", you would be promising that all names added in the future
would look like the examples (for some string of lower case letters
"subsys"):

 subsys.foo
 subsys.bar_baz

or for cgroup infrastructure names (using "kern" or "groups" prefix,
I don't have a clear preference):

 groups.stuff
 groups.this_and_that

Then for instance any name (not already in use) that had either (1) no
embedded dot '.', or (2) at least one character other than "[a-z_.]+",
or (3) other variants too numerous to list, would be safe for user
created group names.

Or, for a simpler and more restrictive regex pattern, don't allow
underscores, resulting in all kernel generated names matching:

 1) the known list, or
 2) "[a-z]+\.[a-z]+"

Note here "safe for user created" names just means safe from collision
with kernel generated names, not necessarily safe from collision with
other user generated names.

That is regardless of what you do here, you cannot protect the
delicate user from possible collision. You can only protect them
from collision with "your" names.

This risks imposing extra complexity on users just so you can avoid
being blamed for the name collisions they might still experience
anyway. When I phrase it that way, my enthusiasm for this proposal
weakens further.

> > And did I say incompatible with released versions?
>
> Not at all incompatible if it requires a mount option to enable it ...

Ah - are you saying I missed another detail? That depending on how they
mounted it, the path might be either of:

/mnt/cgroup/groups/user_created_groupname1/groups/user_created_groupname2
or
/mnt/cgroup/user_created_groupname1/user_created_groupname2

So code that knows something about these paths has to work with either
form (since not all code using these paths will control the mount
relevant option.)

I hope I misunderstood something here.

> That leaves a bit of an ugly taste in my mouth ...

Could you spell out the key reason -you- find it distasteful, perhaps
for a stronger pattern such as "[a-z]+\.[a-z]+" I consider above?
Perhaps I'm missing some reason to share in your revulsion.

> but if people seem to prefer that approach we can go for it.

So long as /dev/cpuset is unscathed, I'm ok either way. Let's
see what others think.

--
I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.940.382.4214

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

---

Subject: Re: [RFC] [PATCH] Re: Prefixing cgroup generic control filenames with "cgroup."
Posted by Paul Menage on Fri, 29 Feb 2008 17:59:00 GMT
View Forum Message <> Reply to Message

On Fri, Feb 29, 2008 at 7:30 AM, Paul Jackson <pj@sgi.com> wrote:
>
> The pattern might be stronger (more restrictive) than "[a-z].*"

>
> The pattern might be something like:
> 1) the known set of existing names (a short, specific list), plus
> 2) "[a-z]+\.[a-z]+(_[a-z]+)*"

Why make it more complex? If we're going for a solution that involves
an implicit partition of the namespace that the user has to be aware
of, then simple is good.

> Note here "safe for user created" names just means safe from collision
> with kernel generated names, not necessarily safe from collision with
> other user generated names.
>
> That is regardless of what you do here, you cannot protect the
> delicate user from possible collision. You can only protect them
> from collision with "your" names.

Of course - I don't think anyone's suggesting that the kernel can do
anything about two competing users fighting over who gets to create
cgroup "Foo". But IMO it's crazy to have multiple uncoordinated
userspace cgroup managers.

> > > And did I say incompatible with released versions?
> >
> > Not at all incompatible if it requires a mount option to enable it ...
>
> Ah - are you saying I missed another detail?

That was one of the questions that I left open - we could add it
defaulting to off, unless a mount option was given.

>
> /mnt/cgroup/groups/user_created_groupname1/groups/user_created_groupname2
> or
>
> /mnt/cgroup/user_created_groupname1/user_created_groupname2

Correct.

>
> So code that knows something about these paths has to work with either
> form (since not all code using these paths will control the mount
> relevant option.)

Yes, but compared to all the other configuration details that a
userspace cgroup manager needs to work with, I think this would be a
minor one.

>
> I hope I misunderstood something here.
>
>
> > That leaves a bit of an ugly taste in my mouth ...
>
> Could you spell out the key reason -you- find it distasteful, perhaps
> for a stronger pattern such as "[a-z]+\.[a-z]+" I consider above?
> Perhaps I'm missing some reason to share in your revulsion.

The main reason it's not my primary choice is that it's an implicit
rule that the user has to know about or risk getting bitten in future.
Making that rule complex is even worse, I think.

Of course it does have the big advantage that no code changes are
needed, just documentation. So if people prefer it, then I'm not going
to fight hard.

Paul

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

---

Subject: Re: [RFC] [PATCH] Re: Prefixing cgroup generic control filenames with "cgroup."
Posted by Paul Jackson on Fri, 29 Feb 2008 19:20:57 GMT
View Forum Message <> Reply to Message

> > 2) "[a-z]+\.[a-z]+(_[a-z]+)*"
>
> Why make it more complex?

It's a trade-off, between how many names the pattern covers,
and how complicated it is. I don't really have a preference
between "[a-z].*", "[a-z]+\.[a-z_]+", or other variants.

This sort of namespace partitioning is common in other venues,
such as one or two leading underscores in various C or Python
names having particular 'system' uses.

Perhaps 90% of users who ever are in a position to construct
a name in the cgroup namespace won't worry much about this one
way or the other. For them, the strlen() of the regex pattern
describing future possible kernel generated cgroup names won't
matter. A few such constructors of cgroup names will appreciate
the precision of whatever rule you create.

> The main reason it's not my primary choice is that it's an implicit
> rule that the user has to know about or risk getting bitten in future.

Ok - as I suspected.

Note again, you can't keep the user from getting bitten. You can
only avoid being one of the biters.

>From the users perspective, since you can't actually eliminate the
risk of them ever seeing a collision, any complexity that you impose
on their code risks being viewed as your taxing them more for your
benefit (avoiding any blame to you) than for their benefit (actually
avoiding all collision risk, which we can't practically do.)

All name spaces co-operatively maintained in the commons have this
risk of collision.

As such name spaces go, the cgroup name space is easy. It's a
small world. A few simple lexical conventions should suffice.

I'd suggest something like you promise to stay in the "[a-z]+\.[a-z_]+"
space, where the leading "[a-z]+" prefix is "cgroup" or one of a
modest, slowly growing, set of cgroup subsystems, plus the existing
grand-fathered names. Nothing here keeps others from intruding in
that same space; you would just be promising not to go outside of
that space.

Self-imposed restraint like this "sells" better. The vast majority can
just pay no mind; the small minority that care will appreciate that
you're imposing constraints on yourself (on cgroup kernel code) that
make their lives a little safer (one less chance of name collision)
without imposing any additional constraints on what they can do or
complexity for which they must inescapably code.

Such restraint is similar to what I see the major users of the similar
cpuset name space doing. They each pick a distinct branding prefix
with which to start the names they add below /dev/cpuset. They do so
without even a suggestion from myself, and without any consultation
amongst themselves; just seems a convenient and sensible thing to do.

--

I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.940.382.4214

Subject: Re: [RFC] Prefixing cgroup generic control filenames with "cgroup."
Posted by Li Zefan on Mon, 03 Mar 2008 07:23:51 GMT
View Forum Message <> Reply to Message

Xpl++ wrote:
> Paul Menage ??????:
>> ...
>>
>> A compromise might be to keep "tasks" unprefixed, and say that future
>> names get the "cgroup." prefix; in this case I'd be inclined to add
>> the prefix to notify_on_release and release_agent on the grounds that
>> there's much less chance of breaking anyone with those files since (I
>> suspect) they're much less used.
>>
> This makes most sense to me. It won't break any existing software
> (most likely) while it seems reasonable to leave 'tasks' unprefixed as
> this is something that any software using any subsystem of cgroup
> would be using anyway and it is not that much associated with a
> particulat subsystem.
>

And it makes most sense to me too, though I still doubt name collision
will be a problem.

Subject: Re: [RFC] Prefixing cgroup generic control filenames with "cgroup."
Posted by Paul Menage on Mon, 03 Mar 2008 08:38:16 GMT
View Forum Message <> Reply to Message

On Thu, Feb 28, 2008 at 2:06 PM, Paul Menage <menage@google.com> wrote:
> On Thu, Feb 28, 2008 at 1:33 PM, <serge@hallyn.com> wrote:
> >
> > You said the set of files belong to cgroup itself is likely to increase
> > - do you have some candidates in mind?
>
> Nothing concrete right now. One example that I already proposed was
> the "cgroup.api" file but that's shelved for now, until such time as I
> actually propose the binary API that it was intended to help support.
>

One likely new file that people agreed a while ago could be useful
would be a "procs" file, similar to "tasks", but acting (and
reporting) on thread groups rather than individual tasks.

Paul

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

---

Subject: Re: [RFC] Prefixing cgroup generic control filenames with "cgroup."
Posted by Paul Menage on Mon, 03 Mar 2008 09:11:51 GMT
View Forum Message <> Reply to Message

So, there have been various options suggested over the course of this thread:

--

1) no code changes, just stake out all names matching a certain regexp
(e.g. "[a-z].*") as being potentially used by the kernel in the
future; document this, and let users who are worried about name
clashes avoid these names

pros: no work involved, avoids potentially complex changes to solve a
possibly non-problem.

cons: leaves an intermingled namespace; since this would be a
convention rather than an enforced rule, users might be unaware that
they're setting themselves up for a fall

--

2) separate out the kernel-generated names and user-generated names by
putting the user-generated names in a "groups" sub-directory (can be a
mount option that's automatically disabled for cpusets).

pros: completely solves problem of intermingled namespaces; makes it
easier to see sub-groups at a glance

cons: extra code, slightly more awkward to deal with in the general
case, is incompatible with the code that was in mainline in the brief
period of time since 2.6.24 was finalized.

--

3) prefix all cgroup-provided files with "cgroup."

pros: hardly any extra code; mostly solves the namespace problem since
user names are much less likely to begin with "cgroup."

cons: changes name of "tasks" file visible in 2.6.24; doesn't help if
a future new subsystem is introduced with a commonly-used name that
might clash with existing user-generated names.

--

4) prefix all future cgroup files with "cgroup." (possibly including
existing notify_on_release and release_agent files?)

pros: no extra code, just involves slightly longer strings for future
new files; no incompatibility issues

cons: ugly inconsistency between new cgroup files and grandfathered
old ones, plus same clash problems as option 3

--

Paul

Subject: Re: [RFC] Prefixing cgroup generic control filenames with "cgroup."
Posted by Balbir Singh on Mon, 03 Mar 2008 09:59:29 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On Thu, Feb 28, 2008 at 2:06 PM, Paul Menage <menage@google.com> wrote:
>> On Thu, Feb 28, 2008 at 1:33 PM, <serge@hallyn.com> wrote:
>> >
>> > You said the set of files belong to cgroup itself is likely to increase
>> > - do you have some candidates in mind?
>>
>> Nothing concrete right now. One example that I already proposed was
>> the "cgroup.api" file but that's shelved for now, until such time as I
>> actually propose the binary API that it was intended to help support.
>>
>
> One likely new file that people agreed a while ago could be useful
> would be a "procs" file, similar to "tasks", but acting (and
> reporting) on thread groups rather than individual tasks.
>

> Paul

Yes, I remember this. This feature would be extremely useful.

--
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

---

Subject: Re: [RFC] Prefixing cgroup generic control filenames with "cgroup."
Posted by Paul Jackson on Wed, 05 Mar 2008 01:24:36 GMT
View Forum Message <> Reply to Message

Paul M wrote (quoted by Li Zefan):
>> A compromise might be to keep "tasks" unprefixed, and say that future
>> names get the "cgroup." prefix;

Another stray idea, for the master of stray ideas ;), how about (1) renaming
all the cgroup files as you like, Paul M, even including tasks to cgroup.tasks
or whatever, and then (2) adding symlinks for the legacy names, such as:

tasks -> cgroup.tasks

--
I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.940.382.4214

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

---