Subject: Which of the virtualization approaches is more suitable for kernel?
Posted by dev on Mon, 20 Feb 2006 15:42:28 GMT
View Forum Message <> Reply to Message

Linus, Andrew,

We need your help on what virtualization approach you would accept to mainstream (if any) and where we should go.

If to drop VPID virtualization which caused many disputes, we actually have the one virtualization solution, but 2 approaches for it. Which one will go depends on the goals and your approval any way.

So what are the approaches?

1. namespaces for all types of resources (Eric W. Biederman)

The proposed solution is similar to fs namespaces. This approach introduces namespaces for IPC, IPv4 networking, IPv6 networking, pids etc. It also adds to task_struct a set of pointers to namespaces which task belongs to, i.e. current->ipv4_ns, current->ipc_ns etc.

Benefits:
- fine grained namespaces
- task_struct points directly to a structure describing the namespace and it's data (not to container)
- maybe a bit more logical/clean implementation, since no effective container is involved unlike a second approach.

Disadvantages:
- it is only proof of concept code right now. nothing more.
- such an approach requires adding of additional argument to many functions (e.g. Eric's patch for networking is 1.5 bigger than openvz). it also adds code for getting namespace from objects. e.g. skb->sk->namespace.
- so it increases stack usage
- it can't efficiently compile to the same not virtualized kernel, which can be undesired for embedded linux.
- fine grained namespaces are actually an obfuscation, since kernel subsystems are tightly interconnected. e.g. network -> sysctl -> proc, mqueues -> netlink, ipc -> fs and most often can be used only as a whole container.
- it involves a bit more complicated procedure of a container create/enter which requires exec or something like this, since there is no effective container which could be simply triggered.

2. containers (OpenVZ.org/linux-vserver.org)

Container solution was discussed before, and actually it is also
namespace solution, but as a whole total namespace, with a single kernel
structure describing it.
Every task has two cotnainer pointers: container and effective
container. The later is used to temporarily switch to other contexts,
e.g. when handling IRQs, TCP/IP etc.

Benefits:
- clear logical bounded container, it is clear when container is alive
and when not.
- it doesn't introduce additional args for most functions, no additional
stack usage.
- it compiles to old good kernel when virtualization if off, so doesn't
disturb other configurations.
- Eric brought an interesting idea about introducing interface like
DEFINE_CPU_VAR(), which could potentially allow to create virtualized
variables automagically and access them via econtainer().
- mature working code exists which is used in production for years, so
first working version can be done much quicker

Disadvantages:
- one additional pointer dereference when accessing virtualized
resources, e.g. current->econtainer->shm_ids

Kirill

Subject: Re: Which of the virtualization approaches is more suitable for kernel?
Posted by Herbert Poetzl on Mon, 20 Feb 2006 16:12:40 GMT
View Forum Message <> Reply to Message

On Mon, Feb 20, 2006 at 06:45:21PM +0300, Kirill Korotaev wrote:
> Linus, Andrew,
>
> We need your help on what virtualization approach you would accept to
> mainstream (if any) and where we should go.
>
> If to drop VPID virtualization which caused many disputes, we actually
> have the one virtualization solution, but 2 approaches for it. Which
> one will go depends on the goals and your approval any way.
>
> So what are the approaches?
>
> 1. namespaces for all types of resources (Eric W. Biederman)
>
> The proposed solution is similar to fs namespaces. This approach
> introduces namespaces for IPC, IPv4 networking, IPv6 networking, pids
> etc. It also adds to task_struct a set of pointers to namespaces which

> task belongs to, i.e. current->ipv4_ns, current->ipc_ns etc.
>
> Benefits:
> - fine grained namespaces
> - task_struct points directly to a structure describing the namespace
> and it's data (not to container)
> - maybe a bit more logical/clean implementation, since no effective
> container is involved unlike a second approach.

> Disadvantages:
> - it is only proof of concept code right now. nothing more.

sorry, but that is no disadvantage in my book ...

> - such an approach requires adding of additional argument to many
> functions (e.g. Eric's patch for networking is 1.5 bigger than openvz).

hmm? last time I checked OpenVZ was quite bloated
compared to Linux-VServer, and Eric's network part
isn't even there yet ...

> it also adds code for getting namespace from objects. e.g.
> skb->sk->namespace.

> - so it increases stack usage

> - it can't efficiently compile to the same not virtualized kernel,
> which can be undesired for embedded linux.

while OpenVZ does?

> - fine grained namespaces are actually an obfuscation, since kernel
> subsystems are tightly interconnected. e.g. network -> sysctl -> proc,
> mqueues -> netlink, ipc -> fs and most often can be used only as a
> whole container.

I think a lot of _strange_ interconnects there could
use some cleanup, and after that the interconenctions
would be very small

> - it involves a bit more complicated procedure of a container
> create/enter which requires exec or something like this, since
> there is no effective container which could be simply triggered.

I don't understand this argument ...

> 2. containers (OpenVZ.org/linux-vserver.org)

please do not generalize here, Linux-VServer does
not use a single container structure as you might
think ...

> Container solution was discussed before, and actually it is also
> namespace solution, but as a whole total namespace, with a single
> kernel structure describing it.

that might be true for OpenVZ, but it is not for
Linux-VServer, as we have structures for network
and process contexts as well as different ones for
disk limits

> Every task has two cotnainer pointers: container and effective
> container. The later is used to temporarily switch to other
> contexts, e.g. when handling IRQs, TCP/IP etc.

this doesn't look very cool to me, as IRQs should
be handled in the host context and TCP/IP in the
proper network space ...

> Benefits:
> - clear logical bounded container, it is clear when container
> is alive and when not.

how does that handle the issues you described with
sockets in wait state which have very long timeouts?

> - it doesn't introduce additional args for most functions,
> no additional stack usage.

a single additional arg here and there won't hurt,
and I'm pretty sure most of them will be in inlined
code, where it doesn't really matter

> - it compiles to old good kernel when virtualization if off,
> so doesn't disturb other configurations.

the question here is, do we really want to turn it
off at all? IMHO the design and implementation
should be sufficiently good so that it does neither
impose unnecessary overhead nor change the default
behaviour ...

> - Eric brought an interesting idea about introducing interface like
> DEFINE_CPU_VAR(), which could potentially allow to create virtualized
> variables automagically and access them via econtainer().

how is that an advantage of the container approach?

> - mature working code exists which is used in production for years,
> so first working version can be done much quicker

from the OpenVZ/Virtuozzo(tm) page:

Specific benefits of Virtuozzo(tm) compared to OpenVZ can be found below:

- Higher VPS density. Virtuozzo(tm) provides efficient memory and file sharing mechanisms enabling higher VPS density and better performance of VPSs.

- Improved Stability, Scalability, and Performance. Virtuozzo(tm)

  on hosts with up-to 32 CPUs.

so I conclude, OpenVZ does not contain the code which
provides all this ...

> Disadvantages:
> - one additional pointer dereference when accessing virtualized
> resources, e.g. current->econtainer->shm_ids

best,
Herbert

> Kirill

Subject: Re: Which of the virtualization approaches is more suitable for kernel?
Posted by dev on Tue, 21 Feb 2006 15:59:16 GMT
View Forum Message <> Reply to Message

>>- such an approach requires adding of additional argument to many
>> functions (e.g. Eric's patch for networking is 1.5 bigger than openvz).
> hmm? last time I checked OpenVZ was quite bloated
> compared to Linux-VServer, and Eric's network part
> isn't even there yet ...
This is rather subjective feeling.
I can tell the same about VServer.

>>- it can't efficiently compile to the same not virtualized kernel,
>> which can be undesired for embedded linux.
> while OpenVZ does?
yes. In _most_ cases does.
If Linus/Andrew/others feel this is not an issues at all I will be the

first who will greet Eric's approach. I'm not against it, though it
looks as a disadvantage for me.

>>- fine grained namespaces are actually an obfuscation, since kernel
>> subsystems are tightly interconnected. e.g. network -> sysctl -> proc,
>> mqueues -> netlink, ipc -> fs and most often can be used only as a
>> whole container.
> I think a lot of _strange_ interconnects there could
> use some cleanup, and after that the interconenctions
> would be very small
Why do you think they are strange!? Is it strange that networking
exports it's sysctls and statictics via proc?
Is it strange for you that IPC uses fs?
It is by _design_.

>>- it involves a bit more complicated procedure of a container
>> create/enter which requires exec or something like this, since
>> there is no effective container which could be simply triggered.
> I don't understand this argument ...
- you need to track dependencies between namespaces (e.g. NAT requires
conntracks, IPC requires FS etc.). this should be handled, otherwise one
container being able to create nested container will be able to make oops.

>>2. containers (OpenVZ.org/linux-vserver.org)
>
>
> please do not generalize here, Linux-VServer does
> not use a single container structure as you might
> think ...
1.
This topic is not a question of single container only...
also AFAIK you use it altogether only.
2.
Just to be clear: once again, I'm not against namespaces.

>>Container solution was discussed before, and actually it is also
>>namespace solution, but as a whole total namespace, with a single
>>kernel structure describing it.
>
> that might be true for OpenVZ, but it is not for
> Linux-VServer, as we have structures for network
> and process contexts as well as different ones for
> disk limits
do you have support for it in tools?
i.e. do you support namespaces somehow? can you create half virtualized
container?

>>Every task has two cotnainer pointers: container and effective

>>container. The later is used to temporarily switch to other
>>contexts, e.g. when handling IRQs, TCP/IP etc.
>
>
> this doesn't look very cool to me, as IRQs should
> be handled in the host context and TCP/IP in the
> proper network space ...
this is exactly what it does.
on IRQ context is switched to host.
In TCP/IP to context of socket or network device.

>>Benefits:
>>- clear logical bounded container, it is clear when container
>> is alive and when not.
> how does that handle the issues you described with
> sockets in wait state which have very long timeouts?
easily.
we have clear logic of container lifetime - it is alive until last
process is alive in it. When processes die, container is destroy and so
does all it's sockets. from namespaces point of view, this means that
lifetime of network namespace is limited to lifetime of pid_namespace.

>>- it doesn't introduce additional args for most functions,
>> no additional stack usage.
> a single additional arg here and there won't hurt,
> and I'm pretty sure most of them will be in inlined
> code, where it doesn't really matter
have you analyzed that before thinking about inlining?

>>- it compiles to old good kernel when virtualization if off,
>> so doesn't disturb other configurations.
> the question here is, do we really want to turn it
> off at all? IMHO the design and implementation
> should be sufficiently good so that it does neither
> impose unnecessary overhead nor change the default
> behaviour ...
this is the question I want to get from Linus/Andrew.
I don't believe in low overhead. It starts from virtualization, then
goes reource management etc.
These features _definetely_ introduce overhead and increase resource
consumption. Not big, but why not configurable?
You don't need CPUSETS on UP i386 machine, do you? Why may I want this
stuff in my embedded Linux? The same for secured Linux distributions,
it only opens the ways for possible security issues.

>>- Eric brought an interesting idea about introducing interface like
>> DEFINE_CPU_VAR(), which could potentially allow to create virtualized
>> variables automagically and access them via econtainer().

> how is that an advantage of the container approach?
Such vars can automatically be defined to something like
"(econtainer()->virtualized_variable)".
This looks similar to percpu variable interfaces.

>>- mature working code exists which is used in production for years,
>> so first working version can be done much quicker
> from the OpenVZ/Virtuozzo(tm) page:
> Specific benefits of Virtuozzo(tm) compared to OpenVZ can be
> found below:
> - Higher VPS density. Virtuozzo(tm) provides efficient memory
>   and file sharing mechanisms enabling higher VPS density and
>   better performance of VPSs.
> - Improved Stability, Scalability, and Performance. Virtuozzo(tm)

>   on hosts with up-to 32 CPUs.
> so I conclude, OpenVZ does not contain the code which
> provides all this ..
:))))
Doesn't provide what? Stability?
Q/A process used for Virtuozzo end up in OpenVZ code eventually as well.
This is more subject of support/QA.
Performance? we optimize systems for customers, have
HA/monitoring/tuning/management tools for it etc.

Seems, you are just trying to move from the topic. Great.

Thanks,
Kirill

Subject: Re: Which of the virtualization approaches is more suitable for kernel?
Posted by Sam Vilain on Tue, 21 Feb 2006 20:33:41 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:
>>>- fine grained namespaces are actually an obfuscation, since kernel
>>> subsystems are tightly interconnected. e.g. network -> sysctl -> proc,
>>> mqueues -> netlink, ipc -> fs and most often can be used only as a
>>> whole container.
>>I think a lot of _strange_ interconnects there could
>>use some cleanup, and after that the interconenctions
>>would be very small
> Why do you think they are strange!? Is it strange that networking
> exports it's sysctls and statictics via proc?
> Is it strange for you that IPC uses fs?
> It is by _design_.

Great, and this kind of simple design also worked well for the first few iterations of Linux-VServer. However, some people need more flexibility as we are seeing by the wide range of virtualisation schemes being proposed. In the 2.1.x VServer patch the network and (process&IPC) isolation and virtualisation have been kept seperate, and can be managed with seperate utilities. There is also a syscall and utility to manage the existing kernel filesystem namespaces.

Eric's pspace work keeps the PID aspect seperate too, which I never envisioned possible.

I think that if we can keep as much seperation between systems as possible, then we will have a cleaner design. Also it will make life easier for the core team as we can more easily divide up the patches for consideration by the relevant subsystem maintainer.

> - you need to track dependencies between namespaces (e.g. NAT requires
> conntracks, IPC requires FS etc.). this should be handled, otherwise one
> container being able to create nested container will be able to make oops.

This is just normal refcounting. Yes, IPC requires filesystem code, but it doesn't care about the VFS, which is what filesystem namespaces abstract.

> do you have support for it in tools?
> i.e. do you support namespaces somehow? can you create half
> virtualized container?

See the util-vserver package, it comes with chbind and vnamespace which allow creation of 'half-virtualized' containers, though most of the rest of the functionality, such as per-vserver ulimits, disklimits, etc have been shoehorned into the general vx_info structure. As we merge into the mainstream we can review each of these decisions and decide whether it is an inherantly per-process decision, or more XX_info structures are warranted.

>>this doesn't look very cool to me, as IRQs should
>>be handled in the host context and TCP/IP in the
>>proper network space ...
> this is exactly what it does.
> on IRQ context is switched to host.
> In TCP/IP to context of socket or network device.

That sounds like an interesting innovation, and we can compare our patches in this space once we have some common terms of reference and starting points.

>>the question here is, do we really want to turn it
>>off at all? IMHO the design and implementation

>>should be sufficiently good so that it does neither
>>impose unnecessary overhead nor change the default
>>behaviour ...
> this is the question I want to get from Linus/Andrew.
> I don't believe in low overhead. It starts from virtualization, then
> goes reource management etc.
> These features _definetely_ introduce overhead and increase resource
> consumption. Not big, but why not configurable?

Obviously, our projects have different goals; Linux-VServer has very
little performance overhead. Special provisions are made to achieve
scalability on SMP and to avoid unnecessary cacheline issues. Once that
is sorted out, it's very hard to measure any performance overhead of it,
especially when the task_struct->vx_info pointer is null.

However I see nothing wrong with making all code disappear without the
kernel config option enabled. I expect that as time goes on, you'd just
as soon disable it as you would disable the open() system call. I think
that's what Herbert was getting at with his comment.

> Seems, you are just trying to move from the topic. Great.

I always did want to be a Lumberjack!

Sam.

Subject: Re: Which of the virtualization approaches is more suitable for kernel?
Posted by Herbert Poetzl on Tue, 21 Feb 2006 23:50:17 GMT
View Forum Message <> Reply to Message

On Tue, Feb 21, 2006 at 07:00:55PM +0300, Kirill Korotaev wrote:
>>>- such an approach requires adding of additional argument to many
>>>functions (e.g. Eric's patch for networking is 1.5 bigger than openvz).
>>hmm? last time I checked OpenVZ was quite bloated
>>compared to Linux-VServer, and Eric's network part
>>isn't even there yet ...
>This is rather subjective feeling.

of course, of course ...

OpenVZ stable patches:
1857829 patch-022stab032-core
1886915 patch-022stab034-core
7390511 patch-022stab045-combined
7570326 patch-022stab050-combined
8042889 patch-022stab056-combined
8059201 patch-022stab064-combined

Linux-VServer stable releases:
  100130 patch-2.4.20-vs1.00.diff
  135068 patch-2.4.21-vs1.20.diff
  587170 patch-2.6.12.4-vs2.0.diff
  593052 patch-2.6.14.3-vs2.01.diff
  619268 patch-2.6.15.4-vs2.0.2-rc6.diff

>I can tell the same about VServer.
really?

>>>- it can't efficiently compile to the same not virtualized kernel,
>>> which can be undesired for embedded linux.
>>while OpenVZ does?
>yes. In _most_ cases does.
>If Linus/Andrew/others feel this is not an issues at all I will be the
>first who will greet Eric's approach. I'm not against it, though it
>looks as a disadvantage for me.
>
>>>- fine grained namespaces are actually an obfuscation, since kernel
>>> subsystems are tightly interconnected. e.g. network -> sysctl -> proc,
>>> mqueues -> netlink, ipc -> fs and most often can be used only as a
>>> whole container.
>>I think a lot of _strange_ interconnects there could
>>use some cleanup, and after that the interconenctions
>>would be very small
>Why do you think they are strange!? Is it strange that networking
>exports it's sysctls and statictics via proc?
>Is it strange for you that IPC uses fs?
>It is by _design_.
>
>>>- it involves a bit more complicated procedure of a container
>>> create/enter which requires exec or something like this, since
>>> there is no effective container which could be simply triggered.
>>I don't understand this argument ...
>- you need to track dependencies between namespaces (e.g. NAT requires
>conntracks, IPC requires FS etc.). this should be handled, otherwise one
>container being able to create nested container will be able to make oops.
>
>>>2. containers (OpenVZ.org/linux-vserver.org)
>>
>>please do not generalize here, Linux-VServer does
>>not use a single container structure as you might
>>think ...
>1.
>This topic is not a question of single container only...
>also AFAIK you use it altogether only.
>2.

>Just to be clear: once again, I'm not against namespaces.
>
>>>Container solution was discussed before, and actually it is also
>>>namespace solution, but as a whole total namespace, with a single
>>>kernel structure describing it.
>>
>>that might be true for OpenVZ, but it is not for
>>Linux-VServer, as we have structures for network
>>and process contexts as well as different ones for
>>disk limits
>do you have support for it in tools?
>i.e. do you support namespaces somehow? can you create half virtualized
>container?

sure, just get the tools and use vnamespace
or if you prefer chcontext or chbind ...

>>>Every task has two cotnainer pointers: container and effective
>>>container. The later is used to temporarily switch to other
>>>contexts, e.g. when handling IRQs, TCP/IP etc.
>>
>>
>>this doesn't look very cool to me, as IRQs should
>>be handled in the host context and TCP/IP in the
>>proper network space ...
>this is exactly what it does.
>on IRQ context is switched to host.
>In TCP/IP to context of socket or network device.
>
>>>Benefits:
>>>- clear logical bounded container, it is clear when container
>>> is alive and when not.
>>how does that handle the issues you described with
>>sockets in wait state which have very long timeouts?
>easily.
>we have clear logic of container lifetime - it is alive until last
>process is alive in it. When processes die, container is destroy and so
>does all it's sockets. from namespaces point of view, this means that
>lifetime of network namespace is limited to lifetime of pid_namespace.
>
>>>- it doesn't introduce additional args for most functions,
>>> no additional stack usage.
>>a single additional arg here and there won't hurt,
>>and I'm pretty sure most of them will be in inlined
>>code, where it doesn't really matter
>have you analyzed that before thinking about inlining?
>
>>>- it compiles to old good kernel when virtualization if off,

>>> so doesn't disturb other configurations.
>>the question here is, do we really want to turn it
>>off at all? IMHO the design and implementation
>>should be sufficiently good so that it does neither
>>impose unnecessary overhead nor change the default
>>behaviour ...
>this is the question I want to get from Linus/Andrew.
>I don't believe in low overhead. It starts from virtualization, then
>goes reource management etc.
>These features _definetely_ introduce overhead and increase resource
>consumption. Not big, but why not configurable?
>You don't need CPUSETS on UP i386 machine, do you? Why may I want this
>stuff in my embedded Linux? The same for secured Linux distributions,
>it only opens the ways for possible security issues.

ah, you got me wrong there, of course embedded
systems have other requirements, and it might
turn out that some of those virtualizations
require config options to disable them ...

but, I do not see a measurable overhead there
and I do not consider it a problem to disable
certain more expensive parts ...

>>>- Eric brought an interesting idea about introducing interface like
>>> DEFINE_CPU_VAR(), which could potentially allow to create virtualized
>>> variables automagically and access them via econtainer().
>>how is that an advantage of the container approach?
>Such vars can automatically be defined to something like
>"(econtainer()->virtualized_variable)".
>This looks similar to percpu variable interfaces.
>
>>>- mature working code exists which is used in production for years,
>>> so first working version can be done much quicker
>>from the OpenVZ/Virtuozzo(tm) page:
>> Specific benefits of Virtuozzo(tm) compared to OpenVZ can be
>> found below:
>> - Higher VPS density. Virtuozzo(tm) provides efficient memory
>>   and file sharing mechanisms enabling higher VPS density and
>>   better performance of VPSs.
>> - Improved Stability, Scalability, and Performance. Virtuozzo(tm)

>>   on hosts with up-to 32 CPUs.
>>so I conclude, OpenVZ does not contain the code which
>>provides all this ..
>:))))
>Doesn't provide what? Stability?
>Q/A process used for Virtuozzo end up in OpenVZ code eventually as well.

>This is more subject of support/QA.
>Performance? we optimize systems for customers, have
>HA/monitoring/tuning/management tools for it etc.
>
>Seems, you are just trying to move from the topic. Great.
I guess I was right on topic ...

best,
Herbert

>Thanks,
>Kirill

Subject: Re: Re: Which of the virtualization approaches is more suitable for kernel?
Posted by kir on Wed, 22 Feb 2006 10:09:23 GMT
View Forum Message <> Reply to Message

Herbert Poetzl wrote:

>On Tue, Feb 21, 2006 at 07:00:55PM +0300, Kirill Korotaev wrote:
>
>
>>>>- such an approach requires adding of additional argument to many
>>>>functions (e.g. Eric's patch for networking is 1.5 bigger than openvz).
>>>>
>>>>
>>>hmm? last time I checked OpenVZ was quite bloated
>>>compared to Linux-VServer, and Eric's network part
>>>isn't even there yet ...
>>>
>>>
>>This is rather subjective feeling.
>>
>>
>
>of course, of course ...
>
>OpenVZ stable patches:
> 1857829 patch-022stab032-core
> 1886915 patch-022stab034-core
> 7390511 patch-022stab045-combined
> 7570326 patch-022stab050-combined
> 8042889 patch-022stab056-combined
> 8059201 patch-022stab064-combined
>
>Linux-VServer stable releases:
> 100130 patch-2.4.20-vs1.00.diff

> 135068 patch-2.4.21-vs1.20.diff
> 587170 patch-2.6.12.4-vs2.0.diff
> 593052 patch-2.6.14.3-vs2.01.diff
> 619268 patch-2.6.15.4-vs2.0.2-rc6.diff
>
>
Herbert,

Please stop seeding, hmm, falseness. OpenVZ patches you mention are
against 2.6.8 kernel, thus they contain tons of backported mainstream
bugfixes and driver updates; so, most of this size is not
virtualization, but general security/stability/drivers stuff. And yes,
that size also indirectly tells how much work we do to keep our users happy.

Back to the topic. If you (or somebody else) wants to see the real size
of things, take a look at broken-out patch set, available from
http://download.openvz.org/kernel/broken-out/. Here (2.6.15-025stab014.1
kernel) we see that it all boils down to:

Virtualization stuff: diff-vemix-20060120-core 817K
Resource management (User Beancounters): diff-ubc-20060120 377K
Two-level disk quota: diff-vzdq-20051219-2 154K

Subject: Re: Re: Which of the virtualization approaches is more suitable for kernel?
Posted by ebiederm on Wed, 22 Feb 2006 15:26:25 GMT
View Forum Message <> Reply to Message

Kir Kolyshkin <kir@openvz.org> writes:

> Please stop seeding, hmm, falseness. OpenVZ patches you mention are against
> 2.6.8 kernel, thus they contain tons of backported mainstream bugfixes and
> driver updates; so, most of this size is not virtualization, but general
> security/stability/drivers stuff. And yes, that size also indirectly tells how
> much work we do to keep our users happy.

I think Herbert was trying to add some balance to the equation.

> Back to the topic. If you (or somebody else) wants to see the real size of
> things, take a look at broken-out patch set, available from
> http://download.openvz.org/kernel/broken-out/. Here (2.6.15-025stab014.1 kernel)
> we see that it all boils down to:

Thanks. This is the first indication I have seen that you even have broken-out
patches. Why those aren't in your source rpms is beyond me. Everything
seems to have been posted in a 2-3 day window at the end of January and the
beginning of February. Is this something you are now providing?

Shakes head. You have a patch in broken-out that is 817K. Do you really maintain it this way as one giant patch?

> Virtualization stuff: diff-vemix-20060120-core 817K
> Resource management (User Beancounters): diff-ubc-20060120 377K
> Two-level disk quota: diff-vzdq-20051219-2 154K

As for the size of my code, sure parts of it are big I haven't really measured. Primarily this is because I'm not afraid of doing the heavy lifting necessary for a clean long term maintainable solution.

Now while all of this is interesting. It really is beside the point because neither the current vserver nor the current openvz code are ready for mainstream kernel inclusion. Please let's not get side tracked playing whose patch is bigger.

Eric

Subject: Re: Re: Which of the virtualization approaches is more suitable for kernel?
Posted by kir on Thu, 23 Feb 2006 12:02:54 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
>>Back to the topic. If you (or somebody else) wants to see the real size of
>>things, take a look at broken-out patch set, available from
>>http://download.openvz.org/kernel/broken-out/. Here (2.6.15-025stab014.1 kernel)
>>we see that it all boils down to:
>
>
> Thanks. This is the first indication I have seen that you even have broken-out
> patches.

When Kirill Korovaev announced OpenVZ patch set on LKML (two times -- initially and for 2.6.15), he gave the links to the broken-out patch set, both times.

> Why those aren't in your source rpms is beyond me.

That reflects our internal organization: we have a core virtualization team which comes up with a core patch (combining all the stuff), and a maintenance team which can add some extra patches (driver updates, some bugfixes). So that extra patches comes up as a separate patches in src.rpms, while virtualization stuff comes up as a single patch. That way it is easier for our maintainters group.

Sure we understand this is not convenient for developers who want to look at our code -- and thus we provide broken-out kernel patch sets

from time to time (not for every release, as it requires some effort
from Kirill, who is really buzy anyway). So, if you want this for a
specific kernel -- just ask.

I understand that this might look strange, but again, this reflects our
internal development structure.

> Everything
> seems to have been posted in a 2-3 day window at the end of January and the
> beginning of February. Is this something you are now providing?

Again, yes, occasionally from time to time, or upon request.

> Shakes head. You have a patch in broken-out that is 817K. Do you really
> maintain it this way as one giant patch?

In that version I took (025stab014) it was indeed as one big patch, and
I believe Kirill maintains it that way.

Previous kernel version (025stab012) was more fine-grained, take a look
at http://download.openvz.org/kernel/broken-out/2.6.15-025stab0 12.1

> Please let's not get side tracked playing whose patch is bigger.

Absolutely agree!

Regards,
  Kir Kolyshkin, OpenVZ team.

Subject: Re: Re: Which of the virtualization approaches is more suitable for kernel?
Posted by ebiederm on Thu, 23 Feb 2006 13:25:26 GMT
View Forum Message <> Reply to Message

Kir Kolyshkin <kir@openvz.org> writes:

> Eric W. Biederman wrote:
>>>Back to the topic. If you (or somebody else) wants to see the real size of
>>>things, take a look at broken-out patch set, available from
>>>http://download.openvz.org/kernel/broken-out/. Here (2.6.15-025stab014.1
> kernel)
>>>we see that it all boils down to:
>> Thanks. This is the first indication I have seen that you even have
>> broken-out patches.
>
> When Kirill Korovaev announced OpenVZ patch set on LKML (two times --
> initially and for 2.6.15), he gave the links to the broken-out patch set, both
> times.

Hmm. I guess I just missed it.

>> Why those aren't in your source rpms is beyond me.
>
> That reflects our internal organization: we have a core virtualization team
> which comes up with a core patch (combining all the stuff), and a maintenance
> team which can add some extra patches (driver updates, some bugfixes). So that
> extra patches comes up as a separate patches in src.rpms, while virtualization
> stuff comes up as a single patch. That way it is easier for our maintainters
> group.
>
> Sure we understand this is not convenient for developers who want to look at our
> code -- and thus we provide broken-out kernel patch sets from time to time (not
> for every release, as it requires some effort from Kirill, who is really buzy
> anyway). So, if you want this for a specific kernel -- just ask.
>
> I understand that this might look strange, but again, this reflects our internal
> development structure.

There is something this brings up. Currently OpenVZ seems to be a
project where you guys do the work and release the source under the
GPL. Making it technically an open source project. However at the
development level it does not appear to be a community project, at
least at the developer level.

There is nothing wrong with not doing involving the larger community
in the development, but what it does mean is that largely as a
developer OpenVZ is uninteresting to me.

>> Shakes head. You have a patch in broken-out that is 817K. Do you really
>> maintain it this way as one giant patch?
>
> In that version I took (025stab014) it was indeed as one big patch, and I
> believe Kirill maintains it that way.
>
> Previous kernel version (025stab012) was more fine-grained, take a look at
> http://download.openvz.org/kernel/broken-out/2.6.15-025stab0 12.1

Looks a little better yes. This actually surprises me because my
past experience is that usually well focused patches are easier
to forward port as they usually have fewer collisions and those
collisions are usually easier to resolve.

Eric

Subject: Re: Re: Which of the virtualization approaches is more suitable for kernel?

Eric W. Biederman wrote:

>>That reflects our internal organization: we have a core virtualization team
>>which comes up with a core patch (combining all the stuff), and a maintenance
>>team which can add some extra patches (driver updates, some bugfixes). So that
>>extra patches comes up as a separate patches in src.rpms, while virtualization
>>stuff comes up as a single patch. That way it is easier for our maintainters
>>group.
>>
>>Sure we understand this is not convenient for developers who want to look at our
>>code -- and thus we provide broken-out kernel patch sets from time to time (not
>>for every release, as it requires some effort from Kirill, who is really buzy
>>anyway). So, if you want this for a specific kernel -- just ask.
>>
>>I understand that this might look strange, but again, this reflects our internal
>>development structure.
>>
>>
>
>There is something this brings up. Currently OpenVZ seems to be a
>project where you guys do the work and release the source under the
>GPL. Making it technically an open source project. However at the
>development level it does not appear to be a community project, at
>least at the developer level.
>
>There is nothing wrong with not doing involving the larger community
>in the development, but what it does mean is that largely as a
>developer OpenVZ is uninteresting to me.
>
>
I though that first thing that makes particular technology interesting
or otherwise appealing to developers is the technology itself, i.e. is
it interesting, appealing, innovative and superior, is it tomorrow today
and so on. From that point, OpenVZ is pretty much interesting. From the
point of openness -- well, you might be right, there's still something
we could do.

I understand it should work both ways -- we should provide easier ways
to access the code, to contribute etc. Still, I see little to no
interest of contributing to OpenVZ kernel. Probably this is because of
high entry level, probably it is because we are not yet open enough or so.

Any way, I would love to hear any comments/suggestions of how we can
improve this situation from our side (and let me express hope you will
improve it from yours:)).

Regards,
Kir.

Subject: Re: Which of the virtualization approaches is more suitable for kernel?
Posted by ebiederm on Fri, 24 Feb 2006 21:44:42 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@sw.ru> writes:

> Linus, Andrew,
>
> We need your help on what virtualization approach you would accept to
> mainstream (if any) and where we should go.
>
> If to drop VPID virtualization which caused many disputes, we actually
> have the one virtualization solution, but 2 approaches for it. Which one
> will go depends on the goals and your approval any way.

My apologies for not replying sooner.

>From the looks of previous replies I think we have some valid commonalities
that we can focus on.

Largely we all agree that to applications things should look exactly as
they do now. Currently we do not agree on management interfaces.

We seem to have much more agreement on everything except pids, so discussing
some of the other pieces looks worth while.

So I propose we the patches to solve the problem into three categories.
- General cleanups that simplify or fix problems now, but have
  a major advantage for our work.
- The kernel internal implementation of the various namespaces
  without an interface to create new ones.
- The new interfaces for how we create and control containers/namesp aces.

This should allow the various approach to start sharing code, getting
progressively closer to each other until we have an implementation
we can agree is ready to go into Linus's kernel. Plus that will
allow us to have our technical flame wars without totally stopping
progress.

We can start on a broad front, looking at several different things.
But I suggest the first thing we all look at is SYSVIPC. It is
currently a clearly recognized namespace in the kernel so the scope is
well defined. SYSVIPC is just complicated enough to have a
non-trivial implementation while at the same time being simple enough

that we can go through the code in exhausting detail. Getting the
group dynamics working properly.

Then we can as a group look at networking, pids, and the other pieces.

But I do think it is important that we take the problem in pieces
because otherwise it is simply to large to review properly.

Eric

Subject: Re: Which of the virtualization approaches is more suitable for kernel?
Posted by Herbert Poetzl on Fri, 24 Feb 2006 23:01:13 GMT
View Forum Message <> Reply to Message

On Fri, Feb 24, 2006 at 02:44:42PM -0700, Eric W. Biederman wrote:
> Kirill Korotaev <dev@sw.ru> writes:
>
> > Linus, Andrew,
> >
> > We need your help on what virtualization approach you would accept
> > to mainstream (if any) and where we should go.
> >
> > If to drop VPID virtualization which caused many disputes, we
> > actually have the one virtualization solution, but 2 approaches for
> > it. Which one will go depends on the goals and your approval any
> > way.
>
> My apologies for not replying sooner.
>
> > From the looks of previous replies I think we have some valid
> > commonalities that we can focus on.
>
> Largely we all agree that to applications things should look exactly
> as they do now. Currently we do not agree on management interfaces.
>
> We seem to have much more agreement on everything except pids, so
> discussing some of the other pieces looks worth while.
>
> So I propose we the patches to solve the problem into three categories.
> - General cleanups that simplify or fix problems now, but have
>   a major advantage for our work.
> - The kernel internal implementation of the various namespaces
>   without an interface to create new ones.
> - The new interfaces for how we create and control containers/namespaces.

proposal accepted on my side

> This should allow the various approach to start sharing code, getting
> progressively closer to each other until we have an implementation we
> can agree is ready to go into Linus's kernel. Plus that will allow us
> to have our technical flame wars without totally stopping progress.
>
> We can start on a broad front, looking at several different things.
> But I suggest the first thing we all look at is SYSVIPC. It is
> currently a clearly recognized namespace in the kernel so the scope is
> well defined. SYSVIPC is just complicated enough to have a non-trivial
> implementation while at the same time being simple enough that we can
> go through the code in exhausting detail. Getting the group dynamics
> working properly.

okay, sounds good ...

> Then we can as a group look at networking, pids, and the other pieces.
>
> But I do think it is important that we take the problem in pieces
> because otherwise it is simply to large to review properly.

definitely

best,
Herbert

> Eric
> -
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
> Please read the FAQ at http://www.tux.org/lkml/

Subject: Re: Which of the virtualization approaches is more suitable for kernel?
Posted by Dave Hansen on Mon, 27 Feb 2006 17:42:12 GMT
View Forum Message <> Reply to Message

On Fri, 2006-02-24 at 14:44 -0700, Eric W. Biederman wrote:
> We can start on a broad front, looking at several different things.
> But I suggest the first thing we all look at is SYSVIPC. It is
> currently a clearly recognized namespace in the kernel so the scope is
> well defined. SYSVIPC is just complicated enough to have a
> non-trivial implementation while at the same time being simple enough
> that we can go through the code in exhausting detail. Getting the
> group dynamics working properly.

Here's a quick stab at the ipc/msg.c portion of this work. The basic
approach was to move msg_ids, msg_bytes, and msg_hdrs into a structure,

put a pointer to that structure in the task_struct and then dynamically
allocate it.

There is still only one system-wide one of these for now. It can
obviously be extended, though. :)

This is a very simple, brute-force, hack-until-it-compiles-and-boots
approach. (I just realized that I didn't check the return of the alloc
properly.)

Is this the form that we'd like these patches to take? Any comments
about the naming? Do we want to keep the _namespace nomenclature, or
does the "context" that I used here make more sense

-- Dave

```
 work-dave/include/linux/ipc.h   |   12 +++
 work-dave/include/linux/sched.h |    1
 work-dave/ipc/msg.c             |  152 ++++++++++++++++++++++++++----------------
 work-dave/ipc/util.c            |    7 +
 work-dave/ipc/util.h            |    2
 work-dave/kernel/fork.c         |    5 +
 6 files changed, 108 insertions(+), 71 deletions(-)

--- work/ipc/msg.c~sysv-container 2006-02-27 09:30:23.000000000 -0800
+++ work-dave/ipc/msg.c 2006-02-27 09:32:18.000000000 -0800
@@ -60,35 +60,44 @@ struct msg_sender {
 #define SEARCH_NOTEQUAL  3
 #define SEARCH_LESSEQUAL 4

-static atomic_t msg_bytes = ATOMIC_INIT(0);
-static atomic_t msg_hdrs = ATOMIC_INIT(0);
+#define msg_lock(ctx, id) ((struct msg_queue*)ipc_lock(&ctx->msg_ids,id))
+#define msg_unlock(ctx, msq) ipc_unlock(&(msq)->q_perm)
+#define msg_rmid(ctx, id) ((struct msg_queue*)ipc_rmid(&ctx->msg_ids,id))
+#define msg_checkid(ctx, msq, msgid) \
+ ipc_checkid(&ctx->msg_ids,&msq->q_perm,msgid)
+#define msg_buildid(ctx, id, seq) \
+ ipc_buildid(&ctx->msg_ids, id, seq)

-static struct ipc_ids msg_ids;
-
-#define msg_lock(id) ((struct msg_queue*)ipc_lock(&msg_ids,id))
-#define msg_unlock(msq) ipc_unlock(&(msq)->q_perm)
-#define msg_rmid(id) ((struct msg_queue*)ipc_rmid(&msg_ids,id))
-#define msg_checkid(msq, msgid) \
- ipc_checkid(&msg_ids,&msq->q_perm,msgid)
-#define msg_buildid(id, seq) \
```

```
- ipc_buildid(&msg_ids, id, seq)
-
-static void freeque (struct msg_queue *msq, int id);
-static int newque (key_t key, int msgflg);
+static void freeque (struct ipc_msg_context *, struct msg_queue *msq, int id);
+static int newque (struct ipc_msg_context *context, key_t key, int id);
 #ifdef CONFIG_PROC_FS
 static int sysvipc_msg_proc_show(struct seq_file *s, void *it);
 #endif

-void __init msg_init (void)
+struct ipc_msg_context *alloc_ipc_msg_context(gfp_t flags)
+{
+ struct ipc_msg_context *msg_context;
+
+ msg_context = kzalloc(sizeof(*msg_context), flags);
+ if (!msg_context)
+  return NULL;
+
+ atomic_set(&msg_context->msg_bytes, 0);
+ atomic_set(&msg_context->msg_hdrs, 0);
+
+ return msg_context;
+}
+
+void __init msg_init (struct ipc_msg_context *context)
 {
- ipc_init_ids(&msg_ids,msg_ctlmni);
+ ipc_init_ids(&context->msg_ids,msg_ctlmni);
  ipc_init_proc_interface("sysvipc/msg",
     "       key      msqid perms      cbytes       qnum lspid lrpid   uid   gid  cuid  cgid      stime      rtime
     ctime\n",
-    &msg_ids,
+    &context->msg_ids,
     sysvipc_msg_proc_show);
 }

-static int newque (key_t key, int msgflg)
+static int newque (struct ipc_msg_context *context, key_t key, int msgflg)
 {
  int id;
  int retval;
@@ -108,14 +117,14 @@ static int newque (key_t key, int msgflg
   return retval;
  }

- id = ipc_addid(&msg_ids, &msq->q_perm, msg_ctlmni);
+ id = ipc_addid(&context->msg_ids, &msq->q_perm, msg_ctlmni);
```

```
  if(id == -1) {
   security_msg_queue_free(msq);
   ipc_rcu_putref(msq);
   return -ENOSPC;
  }

- msq->q_id = msg_buildid(id,msq->q_perm.seq);
+ msq->q_id = msg_buildid(context,id,msq->q_perm.seq);
  msq->q_stime = msq->q_rtime = 0;
  msq->q_ctime = get_seconds();
  msq->q_cbytes = msq->q_qnum = 0;
@@ -124,7 +133,7 @@ static int newque (key_t key, int msgflg
  INIT_LIST_HEAD(&msq->q_messages);
  INIT_LIST_HEAD(&msq->q_receivers);
  INIT_LIST_HEAD(&msq->q_senders);
- msg_unlock(msq);
+ msg_unlock(context, msq);

  return msq->q_id;
 }
@@ -182,23 +191,24 @@ static void expunge_all(struct msg_queue
  * msg_ids.sem and the spinlock for this message queue is hold
  * before freeque() is called. msg_ids.sem remains locked on exit.
  */
-static void freeque (struct msg_queue *msq, int id)
+static void freeque (struct ipc_msg_context *context,
+       struct msg_queue *msq, int id)
 {
  struct list_head *tmp;

  expunge_all(msq,-EIDRM);
  ss_wakeup(&msq->q_senders,1);
- msq = msg_rmid(id);
- msg_unlock(msq);
+ msq = msg_rmid(context, id);
+ msg_unlock(context, msq);

  tmp = msq->q_messages.next;
  while(tmp != &msq->q_messages) {
   struct msg_msg* msg = list_entry(tmp,struct msg_msg,m_list);
   tmp = tmp->next;
-  atomic_dec(&msg_hdrs);
+  atomic_dec(&context->msg_hdrs);
   free_msg(msg);
  }
- atomic_sub(msq->q_cbytes, &msg_bytes);
+ atomic_sub(msq->q_cbytes, &context->msg_bytes);
  security_msg_queue_free(msq);
```

```
  ipc_rcu_putref(msq);
 }
@@ -207,32 +217,34 @@ asmlinkage long sys_msgget (key_t key, i
 {
  int id, ret = -EPERM;
  struct msg_queue *msq;
-
- down(&msg_ids.sem);
+ struct ipc_msg_context *context = current->ipc_msg_context;
+
+ down(&context->msg_ids.sem);
  if (key == IPC_PRIVATE)
-  ret = newque(key, msgflg);
- else if ((id = ipc_findkey(&msg_ids, key)) == -1) { /* key not used */
+  ret = newque(context, key, msgflg);
+ else if ((id = ipc_findkey(&context->msg_ids, key)) == -1) {
+  /* key not used */
   if (!(msgflg & IPC_CREAT))
    ret = -ENOENT;
   else
-   ret = newque(key, msgflg);
+   ret = newque(context, key, msgflg);
  } else if (msgflg & IPC_CREAT && msgflg & IPC_EXCL) {
   ret = -EEXIST;
  } else {
-  msq = msg_lock(id);
+  msq = msg_lock(context, id);
   if(msq==NULL)
    BUG();
   if (ipcperms(&msq->q_perm, msgflg))
    ret = -EACCES;
   else {
-   int qid = msg_buildid(id, msq->q_perm.seq);
+   int qid = msg_buildid(context, id, msq->q_perm.seq);
       ret = security_msg_queue_associate(msq, msgflg);
    if (!ret)
     ret = qid;
   }
-  msg_unlock(msq);
+  msg_unlock(context, msq);
  }
- up(&msg_ids.sem);
+ up(&context->msg_ids.sem);
  return ret;
 }

@@ -333,6 +345,7 @@ asmlinkage long sys_msgctl (int msqid, i
  struct msg_queue *msq;
```

```
  struct msq_setbuf setbuf;
  struct kern_ipc_perm *ipcp;
+ struct ipc_msg_context *context = current->ipc_msg_context;

  if (msqid < 0 || cmd < 0)
   return -EINVAL;
@@ -362,18 +375,18 @@ asmlinkage long sys_msgctl (int msqid, i
   msginfo.msgmnb = msg_ctlmnb;
   msginfo.msgssz = MSGSSZ;
   msginfo.msgseg = MSGSEG;
-  down(&msg_ids.sem);
+  down(&context->msg_ids.sem);
   if (cmd == MSG_INFO) {
-   msginfo.msgpool = msg_ids.in_use;
-   msginfo.msgmap = atomic_read(&msg_hdrs);
-   msginfo.msgtql = atomic_read(&msg_bytes);
+   msginfo.msgpool = context->msg_ids.in_use;
+   msginfo.msgmap = atomic_read(&context->msg_hdrs);
+   msginfo.msgtql = atomic_read(&context->msg_bytes);
   } else {
    msginfo.msgmap = MSGMAP;
    msginfo.msgpool = MSGPOOL;
    msginfo.msgtql = MSGTQL;
   }
-  max_id = msg_ids.max_id;
-  up(&msg_ids.sem);
+  max_id = context->msg_ids.max_id;
+  up(&context->msg_ids.sem);
   if (copy_to_user (buf, &msginfo, sizeof(struct msginfo)))
    return -EFAULT;
   return (max_id < 0) ? 0: max_id;
@@ -385,20 +398,21 @@ asmlinkage long sys_msgctl (int msqid, i
   int success_return;
   if (!buf)
    return -EFAULT;
-  if(cmd == MSG_STAT && msqid >= msg_ids.entries->size)
+  if(cmd == MSG_STAT && msqid >= context->msg_ids.entries->size)
    return -EINVAL;

   memset(&tbuf,0,sizeof(tbuf));

-  msq = msg_lock(msqid);
+  msq = msg_lock(context, msqid);
   if (msq == NULL)
    return -EINVAL;

   if(cmd == MSG_STAT) {
-   success_return = msg_buildid(msqid, msq->q_perm.seq);
```

```
+   success_return =
+    msg_buildid(context, msqid, msq->q_perm.seq);
   } else {
    err = -EIDRM;
-   if (msg_checkid(msq,msqid))
+   if (msg_checkid(context,msq,msqid))
     goto out_unlock;
    success_return = 0;
   }
@@ -419,7 +433,7 @@ asmlinkage long sys_msgctl (int msqid, i
   tbuf.msg_qbytes = msq->q_qbytes;
   tbuf.msg_lspid  = msq->q_lspid;
   tbuf.msg_lrpid  = msq->q_lrpid;
-  msg_unlock(msq);
+  msg_unlock(context, msq);
   if (copy_msqid_to_user(buf, &tbuf, version))
    return -EFAULT;
   return success_return;
@@ -438,14 +452,14 @@ asmlinkage long sys_msgctl (int msqid, i
   return  -EINVAL;
  }

- down(&msg_ids.sem);
- msq = msg_lock(msqid);
+ down(&context->msg_ids.sem);
+ msq = msg_lock(context, msqid);
  err=-EINVAL;
  if (msq == NULL)
   goto out_up;

  err = -EIDRM;
- if (msg_checkid(msq,msqid))
+ if (msg_checkid(context,msq,msqid))
   goto out_unlock_up;
  ipcp = &msq->q_perm;
  err = -EPERM;
@@ -480,22 +494,22 @@ asmlinkage long sys_msgctl (int msqid, i
    * due to a larger queue size.
    */
   ss_wakeup(&msq->q_senders,0);
-  msg_unlock(msq);
+  msg_unlock(context, msq);
   break;
  }
  case IPC_RMID:
-  freeque (msq, msqid);
+  freeque (context, msq, msqid);
   break;
```

```
  }
  err = 0;
 out_up:
- up(&msg_ids.sem);
+ up(&context->msg_ids.sem);
  return err;
 out_unlock_up:
- msg_unlock(msq);
+ msg_unlock(context, msq);
  goto out_up;
 out_unlock:
- msg_unlock(msq);
+ msg_unlock(context, msq);
  return err;
 }

@@ -558,7 +572,8 @@ asmlinkage long sys_msgsnd (int msqid, s
  struct msg_msg *msg;
  long mtype;
  int err;
-
+ struct ipc_msg_context *context = current->ipc_msg_context;
+
  if (msgsz > msg_ctlmax || (long) msgsz < 0 || msqid < 0)
   return -EINVAL;
  if (get_user(mtype, &msgp->mtype))
@@ -573,13 +588,13 @@ asmlinkage long sys_msgsnd (int msqid, s
  msg->m_type = mtype;
  msg->m_ts = msgsz;

- msq = msg_lock(msqid);
+ msq = msg_lock(context, msqid);
  err=-EINVAL;
  if(msq==NULL)
   goto out_free;

  err= -EIDRM;
- if (msg_checkid(msq,msqid))
+ if (msg_checkid(context,msq,msqid))
   goto out_unlock_free;

  for (;;) {
@@ -605,7 +620,7 @@ asmlinkage long sys_msgsnd (int msqid, s
   }
   ss_add(msq, &s);
   ipc_rcu_getref(msq);
-  msg_unlock(msq);
+  msg_unlock(context, msq);
```

```
   schedule();

   ipc_lock_by_ptr(&msq->q_perm);
@@ -630,15 +645,15 @@ asmlinkage long sys_msgsnd (int msqid, s
   list_add_tail(&msg->m_list,&msq->q_messages);
   msq->q_cbytes += msgsz;
   msq->q_qnum++;
-  atomic_add(msgsz,&msg_bytes);
-  atomic_inc(&msg_hdrs);
+  atomic_add(msgsz,&context->msg_bytes);
+  atomic_inc(&context->msg_hdrs);
  }

  err = 0;
  msg = NULL;

 out_unlock_free:
- msg_unlock(msq);
+ msg_unlock(context, msq);
 out_free:
  if(msg!=NULL)
   free_msg(msg);
@@ -670,17 +685,18 @@ asmlinkage long sys_msgrcv (int msqid, s
  struct msg_queue *msq;
  struct msg_msg *msg;
  int mode;
+ struct ipc_msg_context *context = current->ipc_msg_context;

  if (msqid < 0 || (long) msgsz < 0)
   return -EINVAL;
  mode = convert_mode(&msgtyp,msgflg);

- msq = msg_lock(msqid);
+ msq = msg_lock(context, msqid);
  if(msq==NULL)
   return -EINVAL;

  msg = ERR_PTR(-EIDRM);
- if (msg_checkid(msq,msqid))
+ if (msg_checkid(context,msq,msqid))
   goto out_unlock;

  for (;;) {
@@ -720,10 +736,10 @@ asmlinkage long sys_msgrcv (int msqid, s
    msq->q_rtime = get_seconds();
    msq->q_lrpid = current->tgid;
    msq->q_cbytes -= msg->m_ts;
-   atomic_sub(msg->m_ts,&msg_bytes);
```

```
-   atomic_dec(&msg_hdrs);
+   atomic_sub(msg->m_ts,&context->msg_bytes);
+   atomic_dec(&context->msg_hdrs);
    ss_wakeup(&msq->q_senders,0);
-   msg_unlock(msq);
+   msg_unlock(context, msq);
    break;
   }
   /* No message waiting. Wait for a message */
@@ -741,7 +757,7 @@ asmlinkage long sys_msgrcv (int msqid, s
    msr_d.r_maxsize = msgsz;
   msr_d.r_msg = ERR_PTR(-EAGAIN);
   current->state = TASK_INTERRUPTIBLE;
-  msg_unlock(msq);
+  msg_unlock(context, msq);

   schedule();

@@ -794,7 +810,7 @@ asmlinkage long sys_msgrcv (int msqid, s
   if (signal_pending(current)) {
    msg = ERR_PTR(-ERESTARTNOHAND);
 out_unlock:
-   msg_unlock(msq);
+   msg_unlock(context, msq);
    break;
   }
  }
diff -puN ipc/msgutil.c~sysv-container ipc/msgutil.c
diff -puN ipc/sem.c~sysv-container ipc/sem.c
diff -puN ipc/shm.c~sysv-container ipc/shm.c
diff -puN ipc/util.c~sysv-container ipc/util.c
--- work/ipc/util.c~sysv-container 2006-02-27 09:30:24.000000000 -0800
+++ work-dave/ipc/util.c 2006-02-27 09:31:43.000000000 -0800
@@ -45,11 +45,14 @@ struct ipc_proc_iface {
  * The various system5 IPC resources (semaphores, messages and shared
  * memory are initialised
  */
-
+
+struct ipc_msg_context *ipc_msg_context;
 static int __init ipc_init(void)
 {
+ ipc_msg_context = alloc_ipc_msg_context(GFP_KERNEL);
+
  sem_init();
- msg_init();
+ msg_init(ipc_msg_context);
  shm_init();
```

```
  return 0;
 }
diff -puN ipc/util.h~sysv-container ipc/util.h
--- work/ipc/util.h~sysv-container 2006-02-27 09:30:24.000000000 -0800
+++ work-dave/ipc/util.h 2006-02-27 09:30:24.000000000 -0800
@@ -12,7 +12,7 @@
 #define SEQ_MULTIPLIER (IPCMNI)

 void sem_init (void);
-void msg_init (void);
+void msg_init (struct ipc_msg_context *context);
 void shm_init (void);

 struct seq_file;
diff -puN include/linux/sched.h~sysv-container include/linux/sched.h
--- work/include/linux/sched.h~sysv-container 2006-02-27 09:30:24.000000000 -0800
+++ work-dave/include/linux/sched.h 2006-02-27 09:30:24.000000000 -0800
@@ -793,6 +793,7 @@ struct task_struct {
  int link_count, total_link_count;
 /* ipc stuff */
  struct sysv_sem sysvsem;
+ struct ipc_msg_context *ipc_msg_context;
 /* CPU-specific state of this task */
  struct thread_struct thread;
 /* filesystem information */
diff -puN include/linux/ipc.h~sysv-container include/linux/ipc.h
--- work/include/linux/ipc.h~sysv-container 2006-02-27 09:30:24.000000000 -0800
+++ work-dave/include/linux/ipc.h 2006-02-27 09:30:24.000000000 -0800
@@ -2,6 +2,9 @@
 #define _LINUX_IPC_H

 #include <linux/types.h>
+#include <linux/kref.h>
+#include <asm/atomic.h>
+#include <asm/semaphore.h>

 #define IPC_PRIVATE ((__kernel_key_t) 0)

@@ -83,6 +86,15 @@ struct ipc_ids {
  struct ipc_id_ary* entries;
 };

+struct ipc_msg_context {
+ atomic_t msg_bytes;
+ atomic_t msg_hdrs;
+
+ struct ipc_ids msg_ids;
+ struct kref count;
```

```
+};
+
+extern struct ipc_msg_context *alloc_ipc_msg_context(gfp_t flags);
 #endif /* __KERNEL__ */

 #endif /* _LINUX_IPC_H */
--- work/kernel/fork.c~sysv-container 2006-02-27 09:30:24.000000000 -0800
+++ work-dave/kernel/fork.c 2006-02-27 09:30:24.000000000 -0800
@@ -1184,6 +1184,11 @@ static task_t *copy_process(unsigned lon
  }
  attach_pid(p, PIDTYPE_TGID, p->tgid);
  attach_pid(p, PIDTYPE_PID, p->pid);
+ { // this extern will go away when we start to dynamically
+   // allocate these, nothing to see here
+  extern struct ipc_msg_context ipc_msg_context;
+  p->ipc_msg_context = current->ipc_msg_context;
+ }

  nr_threads++;
  total_forks++;
_
```

Subject: Re: Which of the virtualization approaches is more suitable for kernel?
Posted by ebiederm on Mon, 27 Feb 2006 21:14:20 GMT
View Forum Message <> Reply to Message

Dave Hansen <haveblue@us.ibm.com> writes:

> On Fri, 2006-02-24 at 14:44 -0700, Eric W. Biederman wrote:
>> We can start on a broad front, looking at several different things.
>> But I suggest the first thing we all look at is SYSVIPC. It is
>> currently a clearly recognized namespace in the kernel so the scope is
>> well defined. SYSVIPC is just complicated enough to have a
>> non-trivial implementation while at the same time being simple enough
>> that we can go through the code in exhausting detail. Getting the
>> group dynamics working properly.
>
> Here's a quick stab at the ipc/msg.c portion of this work. The basic
> approach was to move msg_ids, msg_bytes, and msg_hdrs into a structure,
> put a pointer to that structure in the task_struct and then dynamically
> allocate it.
>
> There is still only one system-wide one of these for now. It can
> obviously be extended, though. :)
>
> This is a very simple, brute-force, hack-until-it-compiles-and-boots
> approach. (I just realized that I didn't check the return of the alloc

> properly.)
>
> Is this the form that we'd like these patches to take? Any comments
> about the naming? Do we want to keep the _namespace nomenclature, or
> does the "context" that I used here make more sense

I think from 10,000 feet the form is about right.

I like the namespace nomenclature. (It can be shorted to _space or _ns).
In part because it shortens well, and in part because it emphasizes that
we are *just* dealing with the names.

You split the resolution at just ipc_msgs. When I really think it should
be everything ipcs deals with.

Performing the assignment inside the tasklist_lock is not something we
want to do in do_fork().

So it looks like a good start. There are a lot of details yet to be filled
in, proc, sysctl, cleanup on namespace release. (We can still provide
the create destroy methods even if we don't hook the up).

I think in this case I would put the actual namespace structure
definition in util.h, and just put a struct ipc_ns in sched.h.
sysvipc is isolated enough that nothing outside of the ipc/
directory needs to know the implementation details.

It probably makes sense to have a statically structure and
to set the pointer initially in init_task.h

Until we reach the point where we can multiple instances that
even removes the need to have a pointer copy in do_fork()
as that happens already as part of the structure copy.

Eric

Subject: Re: Which of the virtualization approaches is more suitable for kernel?
Posted by Dave Hansen on Mon, 27 Feb 2006 21:35:45 GMT
View Forum Message <> Reply to Message

On Mon, 2006-02-27 at 14:14 -0700, Eric W. Biederman wrote:
> I like the namespace nomenclature. (It can be shorted to _space or _ns).
> In part because it shortens well, and in part because it emphasizes that
> we are *just* dealing with the names.

When I was looking at this, I was pretending to be just somebody looking

at sysv code, with no knowledge of containers or namespaces.

For a person like that, I think names like _space or _ns are pretty much
not an option, unless those terms become as integral to the kernel as
things like kobjects.

> You split the resolution at just ipc_msgs. When I really think it should
> be everything ipcs deals with.

This was just the first patch. :)

> Performing the assignment inside the tasklist_lock is not something we
> want to do in do_fork().

Any particular reason why? There seem to be a number of things done in
there that aren't _strictly_ needed under the tasklist_lock. Where
would you do it?

> So it looks like a good start. There are a lot of details yet to be filled
> in, proc, sysctl, cleanup on namespace release. (We can still provide
> the create destroy methods even if we don't hook the up).

Yeah, I saved shm for last because it has the largest number of outside
interactions. My current thoughts are that we'll need _contexts or
_namespaces associated with /proc mounts as well.

> I think in this case I would put the actual namespace structure
> definition in util.h, and just put a struct ipc_ns in sched.h.

Ahhh, as in

 struct ipc_ns;

And just keep a pointer from the task? Yeah, that does keep it quite
isolated.

-- Dave

Subject: Re: Which of the virtualization approaches is more suitable for kernel?
Posted by ebiederm on Mon, 27 Feb 2006 21:56:37 GMT
View Forum Message <> Reply to Message

Dave Hansen <haveblue@us.ibm.com> writes:

> On Mon, 2006-02-27 at 14:14 -0700, Eric W. Biederman wrote:
>> I like the namespace nomenclature. (It can be shorted to _space or _ns).
>> In part because it shortens well, and in part because it emphasizes that

>> we are *just* dealing with the names.
>
> When I was looking at this, I was pretending to be just somebody looking
> at sysv code, with no knowledge of containers or namespaces.
>
> For a person like that, I think names like _space or _ns are pretty much
> not an option, unless those terms become as integral to the kernel as
> things like kobjects.

To be clear I was talking name suffixes. So ipc_space certainly conveys
something, and even ipc_ns may be ok.

>> You split the resolution at just ipc_msgs. When I really think it should
>> be everything ipcs deals with.
>
> This was just the first patch. :)

:)

Just wanted to make certain we agreed on the scope.

>> Performing the assignment inside the tasklist_lock is not something we
>> want to do in do_fork().
>
> Any particular reason why? There seem to be a number of things done in
> there that aren't _strictly_ needed under the tasklist_lock. Where
> would you do it?

Well all of the other things we can share or not share are already
outside of the tasklist_lock.

We may not be quite minimal but we actually are fairly close to minimal
inside the tasklist_lock.

>> So it looks like a good start. There are a lot of details yet to be filled
>> in, proc, sysctl, cleanup on namespace release. (We can still provide
>> the create destroy methods even if we don't hook the up).
>
> Yeah, I saved shm for last because it has the largest number of outside
> interactions. My current thoughts are that we'll need _contexts or
> _namespaces associated with /proc mounts as well.

Yes. I think the easy way to handle this is to have a symlink
from /proc/sysvipc to /proc/self/sysvipc. And then we have a per
process reporting area.

That preserves all of the old programs but enables us to get the
information out.

>> I think in this case I would put the actual namespace structure
>> definition in util.h, and just put a struct ipc_ns in sched.h.
>
> Ahhh, as in
>
> struct ipc_ns;
>
> And just keep a pointer from the task? Yeah, that does keep it quite
> isolated.

Yep.

Eric
