Subject: Re: [RFC] [PATCH 0/7] Some basic vserver infrastructure
Posted by Sam Vilain on Tue, 21 Mar 2006 23:12:31 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:

>>This patch is simple, but does not handle SMP scalability very well
>>(you'll get a lot of cacheline problems when you start actually using
>>the container structure; the hashing helps a lot there)
>>
>>
>Could you elaborate a bit on this one? What has cacheline problems?
>
>

OK, on reflection this probably doesn't belong this early in the
series. It only helps speed up lookup by IDs which is only an SMP
speed enhancement if you expect a lot of those (eg, when storing the
XIDs on another subsystem, such as a filesystem) and unusual things like
creating/destroying a lot of vservers quickly. I'll pull it out.

I tried to make it as basic as possible, but my basic problem with the
patch you linked to is that it shouldn't be so small that you don't get
a decent bunch of the internal API functions listed in description of
part 1 of the set
(http://vserver.utsl.gen.nz/patches/utsl/2.6.16-rc4-vsi/01-VS erver-Umbrella.diff)

>>and does not provide functions such as looking up a container by ID etc.
>>
>>
>
>We need something so simple that we probably don't even deal with ids.
>I believe that Eric claims that we don't really need container _ids_.
>For instance, the filesystem namespaces have no ids, and work just fine.
>
>

For actual namespace structures themselves that are virtualising real
things, I agree entirely.

But I think the whole point of this patchset (which sadly vger ate for
the wider audience, and I still don't know why) is to group tasks and to
give userland, and hence the administrator, something tangible with
which to group processes with and tack namespace structures onto. I can
split out the ID related stuff into another patch, but it would need to
go back in before a useful syscall interface can be added. I'll
consider it anyway, though.

Note that a given vx_info structure still might share namespace
structures with other vx_info objects - this is what I was alluding to
with the "we haven't made any restrictions on the nature of
virtualisation yet" comment. All we've made is the (XID, PID) tuple
(although the default for XID=0 is that all processes are visible in one
big PID space). The intention is that flags will control how you want
your PIDs to work for that vserver.

The only thing that they can't do is share processes - it is a one to
many relationship. However, if there can be a parent/child
containership relationship on the XIDs themselves, you can still achieve
the behaviour of these unusual situations. I'm working on the
assumption that if we ever have to migrate trees of XIDs then we can
virtualise how they look inside a vserver using flags.

Eric, perhaps you can comment or refer to the earlier post where you
made this argument. I tried to follow it but perhaps I missed the jist
of one of the messages or one of the most important messages entirely.

>By starting painfully simply, we can build on complexity in bits, and
>justify it as we go.
>
>

This is my aim too! I'll keep chopping and changing.

>>And also because the acronym "vx" makes the API look nice, at least to
>>mine and Herbert's eyes, then when you go to the network virtualisation
>>you get "nx_info", etc. However I'm thinking any of these terms might
>>also be right:
>>
>> - "vserver" spelt in full
>> - family
>> - container
>> - jail
>> - task_ns (sort for namespace)
>>
>>Perhaps we can get a ruling from core team on this one, as it's
>>aesthetics :-).
>>
>>
>
>I was in a meeting with a few coworkers, and we were arguing a bit about
>naming. One person there was a manager-type who didn't have any direct
>involvement in the project. We asked him which naming was more clear.
>
>We need to think a bit like that. What is more clear to somebody who
>has never read the code? (Hint "vx_" means nothing. :)

>
>

OK, so let's look at all of the various names that stem from the use of the term "XID" and try to come up with a good naming system for each. I invite the OpenVZ team, Eric and anyone else to put forward their names as well.

Sorry if this is a bit long. Again I invite anyone at all to come forward with a preference or list another set of alternatives. Let's nail this one down, it comes up every time any patchset like this is put forward.

Linux-VServer:

CONFIG_VSERVER - config option
typedef unsigned int xid_t;
struct vx_info;
task_struct->vx_info
task_struct->xid
vx_task_xid(struct task*) - get an XID from a task_struct
vx_current_xid - get XID for current
vx_info_state(struct vx_info*, VXS_FOO) - does vx_info have state
create_vx_info - creates a new context and "hashes" it
lookup_vx_info - lookup a vx_info by xid
get_vx_info - increase refcount of a vx_info
  [...]
release_vx_info - decrease the process count for a vx_info
task_get_vx_info - like get_vx_info, but by process
vx_migrate_task - join task to a vx_info
vxlprintk - debugging printk (for CONFIG_VSERVER_DEBUG)
vxh_alloc_vx_info - history tracing (for CONFIG_VSERVER_HISTORY)
constants:
  VXS_FOO - state bits
  VXF_FOO - vserver flags (to select features)
  VXC_FOO - vserver-specific capabilities
  VCMD_get_version - vserver subcommand names
  VCI_VERSION - perhaps the legacy of this one should die.

Using the term "vserver" and ID term "vsid":

CONFIG_VSERVER - config option
typedef unsigned int vsid_t;
struct vserver
task_struct->vserver
task_struct->vsid
vserver_task_vsid(struct task*) - get an VSID from a task_struct
vserver_current_vsid - get VSID for current

vserver_state(struct vserver*, VS_STATE_FOO) - does vserver hav...
create_vserver - creates a new context and "hashes" it
lookup_vserver - lookup a vserver by vsid
get_vserver - increase refcount of a vserver
[...]
release_vserver - decrease the process count for a vserver
task_get_vserver - like get_vserver, but by process
vserver_migrate_task - join task to a vserver
vserver_debug - debugging printk (for CONFIG_VSERVER_DEBUG)
vserver_hist_alloc_vserver - history tracing (for CONFIG_VSERVER...
constants:
VS_STATE_FOO - state bits
VS_FLAG_FOO - vserver flags (to select features)
VS_CAP_FOO - vserver-specific capabilities
VS_CMD_get_version - vserver subcommand names
VS_VCI_VERSION

Using the term "container" and ID term "cid":

CONFIG_CONTAINERS - config option
typedef unsigned int cid_t;
struct container
task_struct->container
task_struct->cid
container_task_cid(struct task*) - get an CID from a task_struct
container_current_cid - get CID for current
container_state(struct container*, CONTAINER_STATE_FOO) - does c...
create_container - creates a new context and "hashes" it
lookup_container - lookup a container by cid
get_container - increase refcount of a container
[...]
release_container - decrease the process count for a container
task_get_container - like get_container, but by process
contain_task - join task to a container
container_debug - debugging printk (for CONFIG_CONTAINER_DEBUG)
container_hist_alloc_container - history tracing (for CONFIG_CON...
constants:
CONTAINER_STATE_FOO - state bits
CONTAINER_FLAG_FOO - container flags (to select features)
CONTAINER_CAP_FOO - container-specific capabilities
CONTAINER_CMD_get_version - container subcommand names
CONTAINER_VCI_VERSION

Using the term "box" and ID term "boxid":

CONFIG_BOXES - config option
typedef unsigned int boxid_t;
struct box

task_struct->box
task_struct->boxid
box_task_boxid(struct task*) - get an BOXID from a task_struct
box_current_boxid - get BOXID for current
box_state(struct box*, BOX_STATE_FOO) - does box have state
create_box - creates a new context and "hashes" it
lookup_box - lookup a box by boxid
get_box - increase refcount of a box
  [...]
release_box - decrease the process count for a box
task_get_box - like get_box, but by process
box_migrate_task - join task to a box
box_printk - debugging printk (for CONFIG_BOXES_DEBUG)
box_hist_alloc_box - history tracing (for CONFIG_BOXES_HISTORY)
constants:
  BOX_STATE_FOO - state bits
  BOX_FLAG_FOO - box flags (to select features)
  BOX_CAP_FOO - box-specific capabilities
  BOX_CMD_get_version - box subcommand names
  BOX_VCI_VERSION

Using the term "family" and ID term "fid":

CONFIG_FAMILY - config option
typedef unsigned int fid_t;
struct family
task_struct->family
task_struct->fid
task_fid(struct task*) - get an FID from a task_struct
family_current_fid - get FID for current
family_state(struct family*, FAMILY_STATE_FOO) - does family hav...
create_family - creates a new context and "hashes" it
lookup_family - lookup a family by fid
get_family - increase refcount of a family
  [...]
release_family - decrease the process count for a family
task_get_family - like get_family, but by process
family_adopt_task - join task to a family
family_printk - debugging printk (for CONFIG_FAMILY_DEBUG)
family_hist_alloc_family - history tracing (for CONFIG_FAMILY_HI...
constants:
  FAMILY_STATE_FOO - state bits
  FAMILY_FLAG_FOO - family flags (to select features)
  FAMILY_CAP_FOO - family-specific capabilities
  FAMILY_CMD_get_version - family subcommand names
  FAMILY_VCI_VERSION

Using the term "task_ns" and ID term "nsid":

CONFIG_TASK_NS - config option
typedef unsigned int nsid_t;
struct task_ns
task_struct->task_ns
task_struct->nsid
task_nsid(struct task*) - get an NSID from a task_struct
current_nsid - get NSID for current
task_ns_state(struct task_ns*, TASK_NS_STATE_FOO) - does task_ns hav...
create_task_ns - creates a new context and "hashes" it
lookup_task_ns - lookup a task_ns by nsid
get_task_ns - increase refcount of a task_ns
  [...]
release_task_ns - decrease the process count for a task_ns
task_get_task_ns - like get_task_ns, but by process
task_ns_migrate_task - join task to a task_ns
task_ns_printk - debugging printk (for CONFIG_TASK_NS_DEBUG)
task_ns_hist_alloc_task_ns - history tracing (for CONFIG_TASK_NS_HI...
constants:
  TASK_NS_STATE_FOO - state bits
  TASK_NS_FLAG_FOO - task_ns flags (to select features)
  TASK_NS_CAP_FOO - task_ns-specific capabilities
  TASK_NS_CMD_get_version - task_ns subcommand names
  TASK_NS_VCI_VERSION

For the record, I like the term "process family" most. It implies the possibility strict grouping, like last name, as well as allowing heirarchies, but of course in our modern, post-nuclear family age, does not imply a fixed nature of anything ;-).

Happy picking.

Sam.

Subject: Re: [RFC] [PATCH 0/7] Some basic vserver infrastructure
Posted by Sam Vilain on Wed, 22 Mar 2006 05:18:04 GMT
View Forum Message <> Reply to Message

Sam Vilain wrote:

>Using the term "task_ns" and ID term "nsid":
>
> CONFIG_TASK_NS - config option
> typedef unsigned int nsid_t;
> struct task_ns
> task_struct->task_ns
> task_struct->nsid

> task_nsid(struct task*) - get an NSID from a task_struct
> current_nsid - get NSID for current
> task_ns_state(struct task_ns*, TASK_NS_STATE_FOO) - does task_ns hav...
> create_task_ns - creates a new context and "hashes" it
> lookup_task_ns - lookup a task_ns by nsid
> get_task_ns - increase refcount of a task_ns
> [...]
> release_task_ns - decrease the process count for a task_ns
> task_get_task_ns - like get_task_ns, but by process
> task_ns_migrate_task - join task to a task_ns
> task_ns_printk - debugging printk (for CONFIG_TASK_NS_DEBUG)
> task_ns_hist_alloc_task_ns - history tracing (for CONFIG_TASK_NS_HI...
> constants:
> TASK_NS_STATE_FOO - state bits
> TASK_NS_FLAG_FOO - task_ns flags (to select features)
> TASK_NS_CAP_FOO - task_ns-specific capabilities
> TASK_NS_CMD_get_version - task_ns subcommand names
> TASK_NS_VCI_VERSION
>
>

One more (apparently suggested by Eric Biederman, though perhaps he had
different ideas about what it would look like)

CONFIG_SPACE - config option
typedef unsigned int space_t;
struct space_info;
task_struct->space
task_struct->space_id
task_space_id(struct task*) - get an SPACE_ID from a task_struct
current_space_id - get SPACE_ID for current
space_info_state(struct space_info*, TASK_SPACE_STATE_FOO) - does ...
create_space - creates a new space and "hashes" it
lookup_space - lookup a space_info by space_id
get_space_info - increase refcount of a space_info
put_space_info - decrease refcount of a space_info
 [...]
grab_space - increase the process count for a space
release_space - decrease the process count for a space
task_get_space_info - like get_space_info, but by process
space_migrate_task - join task to a space
space_printk - debugging printk (for CONFIG_SPACE_DEBUG)
space_hist_alloc_space - history tracing (for CONFIG_SPACE_HI...
constants:
 SPACE_STATE_FOO - state bits
 SPACE_FLAG_FOO - task_ns flags (to select features)
 SPACE_CAP_FOO - task_ns-specific capabilities
 SPACE_CMD_get_version - task_ns subcommand names

SPACE_SYSCALL_VERSION

Something like that, anyway. I must admit "Task Spaces" sounds a little
less dorky than "Task Namespaces", but doesn't roll off the tongue that
well because of the '-sk s..' combination.

Anyone?

Sam.

Subject: Re: [RFC] [PATCH 0/7] Some basic vserver infrastructure
Posted by ebiederm on Wed, 22 Mar 2006 07:13:49 GMT
View Forum Message <> Reply to Message

Sam Vilain <sam@vilain.net> writes:

> Dave Hansen wrote:
>
>>>This patch is simple, but does not handle SMP scalability very well
>>>(you'll get a lot of cacheline problems when you start actually using
>>>the container structure; the hashing helps a lot there)
>>>
>>>
>>Could you elaborate a bit on this one? What has cacheline problems?
>>
>>
>
> OK, on reflection this probably doesn't belong this early in the
> series. It only helps speed up lookup by IDs which is only an SMP
> speed enhancement if you expect a lot of those (eg, when storing the
> XIDs on another subsystem, such as a filesystem) and unusual things like
> creating/destroying a lot of vservers quickly. I'll pull it out.
>
> I tried to make it as basic as possible, but my basic problem with the
> patch you linked to is that it shouldn't be so small that you don't get
> a decent bunch of the internal API functions listed in description of
> part 1 of the set
> (http://vserver.utsl.gen.nz/patches/utsl/2.6.16-rc4-vsi/01-VS erver-Umbrella.diff)

Right. I think the smallest thing that we can reasonably discuss with
real problems is the sysvipc namespace. This is why I suggested efforts
in that direction.

>>>and does not provide functions such as looking up a container by ID etc.
>>>
>>>

>>
>>We need something so simple that we probably don't even deal with ids.
>>I believe that Eric claims that we don't really need container _ids_.
>>For instance, the filesystem namespaces have no ids, and work just fine.
>>
>>
>
> For actual namespace structures themselves that are virtualising real
> things, I agree entirely.
>
> But I think the whole point of this patchset (which sadly vger ate for
> the wider audience, and I still don't know why) is to group tasks and to
> give userland, and hence the administrator, something tangible with
> which to group processes with and tack namespace structures onto. I can
> split out the ID related stuff into another patch, but it would need to
> go back in before a useful syscall interface can be added. I'll
> consider it anyway, though.

So as best I can determine that is simply an implementation optimization.
(That is with a little refactoring we can add a structure like that later
 if the performance concerns warrant it.)

Skipping that optimization we should be able to concentrate on the
fundamentals. Which should be simpler and allow better forward progress.

> Note that a given vx_info structure still might share namespace
> structures with other vx_info objects - this is what I was alluding to
> with the "we haven't made any restrictions on the nature of
> virtualisation yet" comment. All we've made is the (XID, PID) tuple
> (although the default for XID=0 is that all processes are visible in one
> big PID space). The intention is that flags will control how you want
> your PIDs to work for that vserver.

Do we need flags or can we move this to user space?

> The only thing that they can't do is share processes - it is a one to
> many relationship. However, if there can be a parent/child
> containership relationship on the XIDs themselves, you can still achieve
> the behaviour of these unusual situations. I'm working on the
> assumption that if we ever have to migrate trees of XIDs then we can
> virtualise how they look inside a vserver using flags.
>
> Eric, perhaps you can comment or refer to the earlier post where you
> made this argument. I tried to follow it but perhaps I missed the jist
> of one of the messages or one of the most important messages entirely.

I'm not certain which argument you are referring to but I will state my general
argument against adding additional global ids.

Ultimately you want to nest you vservers and the like inside of each
other, because if you haven't captured everything you can do in user
space there will be some applications that don't work, and ultimately
it is desirable to support all applications. When doing that you want
to use the same mechanism on the inside as you have on the outside.
Additional global identifiers make this very difficult.

You can always talk about resources by specifying the processes that
use them. It is one level of indirection, but it works and is simple.

>>By starting painfully simply, we can build on complexity in bits, and
>>justify it as we go.
>>
> This is my aim too! I'll keep chopping and changing.
>
>>>And also because the acronym "vx" makes the API look nice, at least to
>>>mine and Herbert's eyes, then when you go to the network virtualisation
>>>you get "nx_info", etc. However I'm thinking any of these terms might
>>>also be right:
>>>
>>> - "vserver" spelt in full
>>> - family
>>> - container
>>> - jail
>>> - task_ns (sort for namespace)
>>>
>>>Perhaps we can get a ruling from core team on this one, as it's
>>>aesthetics :-).
>>>
>>>
>>
>>I was in a meeting with a few coworkers, and we were arguing a bit about
>>naming. One person there was a manager-type who didn't have any direct
>>involvement in the project. We asked him which naming was more clear.
>>
>>We need to think a bit like that. What is more clear to somebody who
>>has never read the code? (Hint "vx_" means nothing. :)
>>
>>
>
> OK, so let's look at all of the various names that stem from the use of
> the term "XID" and try to come up with a good naming system for each. I
> invite the OpenVZ team, Eric and anyone else to put forward their names
> as well.
>
> Sorry if this is a bit long. Again I invite anyone at all to come

> forward with a preference or list another set of alternatives. Let's
> nail this one down, it comes up every time any patchset like this is put
> forward.

> For the record, I like the term "process family" most. It implies the
> possibility strict grouping, like last name, as well as allowing
> heirarchies, but of course in our modern, post-nuclear family age, does
> not imply a fixed nature of anything ;-).

Families don't sound bad, but this all feels like putting the cart
before the horse. It is infrastructure solving a problem that I am
not at all certain is interesting.

Eric

Subject: Re: [RFC] [PATCH 0/7] Some basic vserver infrastructure
Posted by Sam Vilain on Thu, 23 Mar 2006 04:17:27 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:

>>I tried to make it as basic as possible, but my basic problem with the
>>patch you linked to is that it shouldn't be so small that you don't get
>>a decent bunch of the internal API functions listed in description of
>>part 1 of the set
>>(http://vserver.utsl.gen.nz/patches/utsl/2.6.16-rc4-vsi/01-VS erver-Umbrella.diff)
>>
>>
>
>Right. I think the smallest thing that we can reasonably discuss with
>real problems is the sysvipc namespace. This is why I suggested efforts
>in that direction.
>
>

I apologise for not contributing to that effort, however I was expecting
to be able to plug the deliverables of it into this framework. I hope to
participate in as many of these subsystem virtualisation discussions as
I can, but at that time it looked like there was enough input going in.

>>But I think the whole point of this patchset (which sadly vger ate for
>>the wider audience, and I still don't know why) is to group tasks and to
>>give userland, and hence the administrator, something tangible with
>>which to group processes with and tack namespace structures onto. I can
>>split out the ID related stuff into another patch, but it would need to
>>go back in before a useful syscall interface can be added. I'll

>>consider it anyway, though.
>>
>>
>So as best I can determine that is simply an implementation optimization.
>(That is with a little refactoring we can add a structure like that later
> if the performance concerns warrant it.)
>
>Skipping that optimization we should be able to concentrate on the
>fundamentals. Which should be simpler and allow better forward progress.
>
>

Sure, this is where we came to last time. I agree that it is an optimisation, but not just an implementation optimisation.

In the abstract sense you have tuples of (task, namespace) for each type of namespace. However for (task, utsname) this is a much weaker thing to want per-process, and much more like you want it per process family. So, it is also a design optimisation. Some tuples really are (family, namespace) in the first instance.

>>Note that a given vx_info structure still might share namespace
>>structures with other vx_info objects - this is what I was alluding to
>>with the "we haven't made any restrictions on the nature of
>>virtualisation yet" comment. All we've made is the (XID, PID) tuple
>>(although the default for XID=0 is that all processes are visible in one
>>big PID space). The intention is that flags will control how you want
>>your PIDs to work for that vserver.
>>
>>
>Do we need flags or can we move this to user space?
>
>

Some of them influence behaviour in fastpath code. For instance, one example of a flag is the scheduling policy. A process family might have a flag set on it to group its scheduling together, to provide assertions like "this family should get ¼ of the CPU". Clearly userspace can't make this call. But perhaps I don't get your meaning.

I think they are a pragmatic necessity.

>>The only thing that they can't do is share processes - it is a one to
>>many relationship. However, if there can be a parent/child
>>containership relationship on the XIDs themselves, you can still achieve
>>the behaviour of these unusual situations. I'm working on the
>>assumption that if we ever have to migrate trees of XIDs then we can
>>virtualise how they look inside a vserver using flags.

>>
>>Eric, perhaps you can comment or refer to the earlier post where you
>>made this argument. I tried to follow it but perhaps I missed the jist
>>of one of the messages or one of the most important messages entirely.
>>
>>
>
>I'm not certain which argument you are referring to but I will state my general
>argument against adding additional global ids.
>
>Ultimately you want to nest you vservers and the like inside of each
>other, because if you haven't captured everything you can do in user
>space there will be some applications that don't work, and ultimately
>it is desirable to support all applications. When doing that you want
>to use the same mechanism on the inside as you have on the outside.
>Additional global identifiers make this very difficult.
>
>

Sure, you might set a flag on a process family to allow family_id values
to be rewritten on the system/user barrier.

>You can always talk about resources by specifying the processes that
>use them. It is one level of indirection, but it works and is simple.
>
>

Sometimes you might not know of any particular processes in the family.
Maybe it's got one process that is just doing "while (1) { fork && exit
}". How do you "catch" it ?

>>For the record, I like the term "process family" most. It implies the
>>possibility strict grouping, like last name, as well as allowing
>>heirarchies, but of course in our modern, post-nuclear family age, does
>>not imply a fixed nature of anything ;-).
>>
>>
>
>Families don't sound bad, but this all feels like putting the cart
>before the horse. It is infrastructure solving a problem that I am
>not at all certain is interesting.
>
>

So, you're breeding and selecting the horse. Fine, can't I start nailing
together the frame for the cart, and tanning the leather for the bridle?
Surely whichever horse we end up with, we'll still need a cart. And
don't forget the stables, too. We've got a lot of stables already, so we

know roughly what size the cart needs to be. Surely we won't want people knocking those down and rebuilding them.

I'm just a bit concerned about this merging effort turning into a research project. I'd rather try and merge the approaches used by systems people are currently using than try to come up with something wildly new.

After all, if the new developments are that flexible, they should be easily able to support the presented interfaces ranging from the circa 1999 FreeBSD jail() to a more modern vserver/container/openvz/etc API.

Sam.

Subject: Re: Re: [RFC] [PATCH 0/7] Some basic vserver infrastructure
Posted by dev on Fri, 24 Mar 2006 15:36:52 GMT
View Forum Message <> Reply to Message

>> I tried to make it as basic as possible, but my basic problem with the
>> patch you linked to is that it shouldn't be so small that you don't get
>> a decent bunch of the internal API functions listed in description of
>> part 1 of the set
>> (http://vserver.utsl.gen.nz/patches/utsl/2.6.16-rc4-vsi/01-VS erver-Umbrella.diff)
>
> Right. I think the smallest thing that we can reasonably discuss with
> real problems is the sysvipc namespace. This is why I suggested efforts
> in that direction.
ok. I send to all on CC our IPC and utsname namespace patch.
We implemented them as separate namespaces (to please Eric), compile
time configurable (to please embedded people) and using "current"
namespace context (as we do in OpenVZ).
I suppose all these points can be disucssed. In many respects the
patches look like a bit changed Eric/DHansen/OpenVZ.
I suppose this should be discussed/worked out and commited to
Linus/Andrew as there are any serious issues here.

>>> We need something so simple that we probably don't even deal with ids.
>>> I believe that Eric claims that we don't really need container _ids_.
>>> For instance, the filesystem namespaces have no ids, and work just fine.
Eric, the namespaces itself can be without any IDs, really.
But on top of that projects like OpenVZ/vserver can implement VPSs,
umbreallas etc. and will introduce IDs and interfaces they are
accustomed of.

> Ultimately you want to nest you vservers and the like inside of each
> other, because if you haven't captured everything you can do in user

> space there will be some applications that don't work, and ultimately
> it is desirable to support all applications. When doing that you want
> to use the same mechanism on the inside as you have on the outside.
> Additional global identifiers make this very difficult.
>
> You can always talk about resources by specifying the processes that
> use them. It is one level of indirection, but it works and is simple.
I think it is not that easy as you wish to think of it.
Consider 2 processes: A and B.
each of them has its own fs namespace, but common network namespace.
process A passes file descriptor fd to process B via unix socket.
So the same file belongs to 2 different fs namespaces.
Will you simply forbid fd passing to another fd namespace?

I see 2 possibilities here.
1. You either introduce fully isolated resource namespaces, than it is
flat, not nested.
2. Or you need to state correctly what "nesting" means from your point
of view and which operations parent namespace can do with its children.
Last time I had a talk with Herbert we ended up that such a nesting is
nothing more than a privileges delegation, e.g. allows you to manage
your children.
Eric, can your describe what you want from nesting on some particular
namespace, for example fs, ipc, networking, ...? Not by word "nesting",
but by concrete operations which parent can do and why a child looks
"nested".

Just to make it more clear: my understanding of word "nested" means that
if you have, for example, a nested IPC namespace, than parent can see
all the resources (sems, shms, ...) of it's children and have some
private, while children see only its own set of private resources. But
it doesn't look like you are going to implement anything like this.
So what is nesting then? Ability to create namespace? To delegate it
some part of own resource limits?

>>>> And also because the acronym "vx" makes the API look nice, at least to
>>>> mine and Herbert's eyes, then when you go to the network virtualisation
>>>> you get "nx_info", etc. However I'm thinking any of these terms might
>>>> also be right:
>>>>
>>>> - "vserver" spelt in full
>>>> - family
>>>> - container
>>>> - jail
>>>> - task_ns (sort for namespace)
>>>>
>>>> Perhaps we can get a ruling from core team on this one, as it's
>>>> aesthetics :-).

I propose to use "namespace" naming.
1. This is already used in fs.
2. This is what IMHO suites at least OpenVZ/Eric
3. it has good acronym "ns".

>> For the record, I like the term "process family" most. It implies the
>> possibility strict grouping, like last name, as well as allowing
>> heirarchies, but of course in our modern, post-nuclear family age, does
>> not imply a fixed nature of anything ;-).
>
> Families don't sound bad, but this all feels like putting the cart
> before the horse. It is infrastructure solving a problem that I am
> not at all certain is interesting.
agreed.

Thanks,
Kirill

Subject: Re: Re: [RFC] [PATCH 0/7] Some basic vserver infrastructure
Posted by serue on Mon, 27 Mar 2006 12:45:17 GMT
View Forum Message <> Reply to Message

Quoting Kirill Korotaev (dev@sw.ru):
> Just to make it more clear: my understanding of word "nested" means that
> if you have, for example, a nested IPC namespace, than parent can see
> all the resources (sems, shms, ...) of it's children and have some
> private, while children see only its own set of private resources. But
> it doesn't look like you are going to implement anything like this.
> So what is nesting then? Ability to create namespace? To delegate it
> some part of own resource limits?

Nesting simply means that any child ns can create child namespaces of
it's own.

In particular, the following scenario should be perfectly valid:

```
Machine 1              Machine 2
  Xen VM1.1              Xen VM2.1
    vserv 1.1.1          vserv2.1.1
      cont1.1.1.1          cont2.1.1.1
      cont1.1.1.2          cont2.1.1.2
      cont1.1.1.n          cont2.1.1.n
    vserv 1.1.2          vserv2.1.2
      cont1.1.2.1          cont2.1.2.1
      cont1.1.2.2          cont2.1.2.2
      cont1.1.2.n          cont2.1.2.n
  Xen VM1.2              Xen VM2.2
```

vserv 1.2.1	vserv2.2.1
cont1.2.1.1	cont2.2.1.1
cont1.2.1.2	cont2.2.1.2
cont1.2.1.n	cont2.2.1.n
vserv 1.2.2	vserv2.2.2
cont1.2.2.1	cont2.2.2.1
cont1.2.2.2	cont2.2.2.2
cont1.2.2.n	cont2.2.2.n

where containers are used for each virtual server and each container,
so that we can migrate entire VMs, entire virtual servers, or any
container.

> >>>>Perhaps we can get a ruling from core team on this one, as it's
> >>>>aesthetics :-).
> I propose to use "namespace" naming.
> 1. This is already used in fs.
> 2. This is what IMHO suites at least OpenVZ/Eric
> 3. it has good acronym "ns".

I agree.

-serge

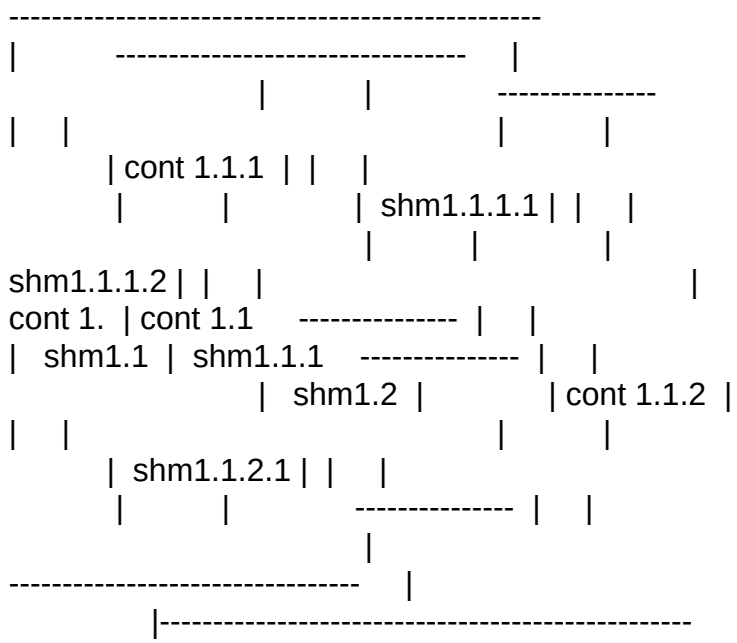
Subject: Re: Re: [RFC] [PATCH 0/7] Some basic vserver infrastructure
Posted by dev on Wed, 29 Mar 2006 12:07:21 GMT
View Forum Message <> Reply to Message

Serge,

Serge E. Hallyn wrote:
> Quoting Kirill Korotaev (dev@sw.ru):
>> Just to make it more clear: my understanding of word "nested" means that
>> if you have, for example, a nested IPC namespace, than parent can see
>> all the resources (sems, shms, ...) of it's children and have some
>> private, while children see only its own set of private resources. But
>> it doesn't look like you are going to implement anything like this.
>> So what is nesting then? Ability to create namespace? To delegate it
>> some part of own resource limits?
>
> Nesting simply means that any child ns can create child namespaces of
> it's own.
your picture below doesn't show that containers have nested containers.
You draw a plain container set inside vserv.
What I mean is that if some container user can create another container,
it DOES not mean it is nested. It is just about permitions to create
other containers. Nested containers in my POV is something different,

when you can see the resources of your container and your children. You see?

I will try to show what I mean on a picture:



You see what I mean? In this example with IPC sharememory container 1 can see all the shm segments. while container1.1.2 can see only his private one smm1.1.2.1.

And if resources are not nested like this, than it is a PLAIN container structure.

Kirill

> In particular, the following scenario should be perfectly valid:
>
> Machine 1 Machine 2
> Xen VM1.1 Xen VM2.1
> vserv 1.1.1 vserv2.1.1
> cont1.1.1.1 cont2.1.1.1
> cont1.1.1.2 cont2.1.1.2
> cont1.1.1.n cont2.1.1.n
> vserv 1.1.2 vserv2.1.2
> cont1.1.2.1 cont2.1.2.1
> cont1.1.2.2 cont2.1.2.2
> cont1.1.2.n cont2.1.2.n
> Xen VM1.2 Xen VM2.2
> vserv 1.2.1 vserv2.2.1
> cont1.2.1.1 cont2.2.1.1
> cont1.2.1.2 cont2.2.1.2
> cont1.2.1.n cont2.2.1.n

> vserv 1.2.2 vserv2.2.2
> cont1.2.2.1 cont2.2.2.1
> cont1.2.2.2 cont2.2.2.2
> cont1.2.2.n cont2.2.2.n
>
> where containers are used for each virtual server and each container,
> so that we can migrate entire VMs, entire virtual servers, or any
> container.
>
>>>>>> Perhaps we can get a ruling from core team on this one, as it's
>>>>>> aesthetics :-).
>> I propose to use "namespace" naming.
>> 1. This is already used in fs.
>> 2. This is what IMHO suites at least OpenVZ/Eric
>> 3. it has good acronym "ns".
>
> I agree.
>
> -serge
>

Subject: Re: Re: [RFC] [PATCH 0/7] Some basic vserver infrastructure
Posted by serue on Wed, 29 Mar 2006 13:47:09 GMT
View Forum Message <> Reply to Message

Quoting Kirill Korotaev (dev@sw.ru):
> Serge,
>
> Serge E. Hallyn wrote:
> >Quoting Kirill Korotaev (dev@sw.ru):
> >>Just to make it more clear: my understanding of word "nested" means that
> >>if you have, for example, a nested IPC namespace, than parent can see
> >>all the resources (sems, shms, ...) of it's children and have some
> >>private, while children see only its own set of private resources. But
> >>it doesn't look like you are going to implement anything like this.
> >>So what is nesting then? Ability to create namespace? To delegate it
> >>some part of own resource limits?
> >
> >Nesting simply means that any child ns can create child namespaces of
> >it's own.
> your picture below doesn't show that containers have nested containers.
> You draw a plain container set inside vserv.

And I am assuming that vserv is implemented as a container, hence this
is an example of nested containers very likely to be used. But given
what I now think is your definition of nested, I think we are agreed.

> What I mean is that if some container user can create another container,
> it DOES not mean it is nested. It is just about permitions to create
> other containers. Nested containers in my POV is something different,
> when you can see the resources of your container and your children. You see?

Alas, the spacing on the picture didn't quite work out :) I think that
by nested containers, you mean overlapping nested containers. In your
example, how are you suggesting that cont1 refers to items in
container1.1.2's shmem? I assume, given your previous posts on openvz,
that you want every shmem id in all namespaces "nested" under cont1 to
be unique, and for cont1 to refer to any item in container1.1.2's
namespace just as it would any of cont1's own shmem?

In that case I am not sure of the actual usefulness. Someone with
different use for containers (you? :) will need to justify it. For me,
pure isolation works just fine. Clearly it will be most useful if we
want fine-grained administration, from parent namespaces, of the items
in a child namespace.

-serge

> I will try to show what I mean on a picture:
>
> ---
> | ---
> | | ---
> | | | |
> | cont 1.1.1 | | | |
> | | | shm1.1.1.1 | | |
> | | | |
> shm1.1.1.2 | | | |
> cont 1. | cont 1.1 --- | |
> | shm1.1 | shm1.1.1 --- | |
> | shm1.2 | | cont 1.1.2 |
> | | | |
> | shm1.1.2.1 | | |
> | | --- | |
> |
> ---
> |---
>
> You see what I mean? In this example with IPC sharememory container 1
> can see all the shm segments. while container1.1.2 can see only his
> private one smm1.1.2.1.
>
> And if resources are not nested like this, than it is a PLAIN container
> structure.
>

> Kirill
>
> >In particular, the following scenario should be perfectly valid:
> >
> > Machine 1 Machine 2
> > Xen VM1.1 Xen VM2.1
> > vserv 1.1.1 vserv2.1.1
> > cont1.1.1.1 cont2.1.1.1
> > cont1.1.1.2 cont2.1.1.2
> > cont1.1.1.n cont2.1.1.n
> > vserv 1.1.2 vserv2.1.2
> > cont1.1.2.1 cont2.1.2.1
> > cont1.1.2.2 cont2.1.2.2
> > cont1.1.2.n cont2.1.2.n
> > Xen VM1.2 Xen VM2.2
> > vserv 1.2.1 vserv2.2.1
> > cont1.2.1.1 cont2.2.1.1
> > cont1.2.1.2 cont2.2.1.2
> > cont1.2.1.n cont2.2.1.n
> > vserv 1.2.2 vserv2.2.2
> > cont1.2.2.1 cont2.2.2.1
> > cont1.2.2.2 cont2.2.2.2
> > cont1.2.2.n cont2.2.2.n
> >
> >where containers are used for each virtual server and each container,
> >so that we can migrate entire VMs, entire virtual servers, or any
> >container.
> >
> >>>>>>Perhaps we can get a ruling from core team on this one, as it's
> >>>>>>aesthetics :-).
> >>I propose to use "namespace" naming.
> >>1. This is already used in fs.
> >>2. This is what IMHO suites at least OpenVZ/Eric
> >>3. it has good acronym "ns".
> >
> >I agree.
> >
> >-serge
> >

Subject: Re: Re: [RFC] [PATCH 0/7] Some basic vserver infrastructure
Posted by Sam Vilain on Wed, 29 Mar 2006 21:30:39 GMT
View Forum Message <> Reply to Message

On Wed, 2006-03-29 at 07:47 -0600, Serge E. Hallyn wrote:
> Alas, the spacing on the picture didn't quite work out :) I think that
> by nested containers, you mean overlapping nested containers. In your

> example, how are you suggesting that cont1 refers to items in
> container1.1.2's shmem? I assume, given your previous posts on openvz,
> that you want every shmem id in all namespaces "nested" under cont1 to
> be unique, and for cont1 to refer to any item in container1.1.2's
> namespace just as it would any of cont1's own shmem?
>
> In that case I am not sure of the actual usefulness. Someone with
> different use for containers (you? :) will need to justify it. For me,
> pure isolation works just fine. Clearly it will be most useful if we
> want fine-grained administration, from parent namespaces, of the items
> in a child namespace.

The overlapping is important if you want to pretend that the
namespace-able resources are allowed to be specified per-process, when
really they are specified per-family.

In this way, a process family is merely a grouping of processes with
like namespaces, and depending on which way they overlap you get the
same behaviour as when processes only have one resource different, and
therefore remove the overhead on fork().

Sam.

Subject: Re: Re: [RFC] [PATCH 0/7] Some basic vserver infrastructure
Posted by ebiederm on Wed, 19 Apr 2006 07:50:33 GMT
View Forum Message <> Reply to Message

Sam Vilain <sam@vilain.net> writes:

> On Wed, 2006-03-29 at 07:47 -0600, Serge E. Hallyn wrote:
>> Alas, the spacing on the picture didn't quite work out :) I think that
>> by nested containers, you mean overlapping nested containers. In your
>> example, how are you suggesting that cont1 refers to items in
>> container1.1.2's shmem? I assume, given your previous posts on openvz,
>> that you want every shmem id in all namespaces "nested" under cont1 to
>> be unique, and for cont1 to refer to any item in container1.1.2's
>> namespace just as it would any of cont1's own shmem?
>>
>> In that case I am not sure of the actual usefulness. Someone with
>> different use for containers (you? :) will need to justify it. For me,
>> pure isolation works just fine. Clearly it will be most useful if we
>> want fine-grained administration, from parent namespaces, of the items
>> in a child namespace.
>
> The overlapping is important if you want to pretend that the
> namespace-able resources are allowed to be specified per-process, when
> really they are specified per-family.

>
> In this way, a process family is merely a grouping of processes with
> like namespaces, and depending on which way they overlap you get the
> same behaviour as when processes only have one resource different, and
> therefore remove the overhead on fork().

I missed this subthread originally.

I think it is important that we can have containers in containers
if at all possible. This means large software collections can count
on them being present.

As for having some items inside a namespace show up in both
a parent and a child namespace I think the case is less clearly
defined. If possible that is something we want to avoid as it
complicates the implementation.

For pids I will be surprised if we can avoid it.

For most other namespaces I think we can, and it is a good thing
to avoid.

Eric

Subject: Re: Re: [RFC] [PATCH 0/7] Some basic vserver infrastructure
Posted by Sam Vilain on Wed, 19 Apr 2006 21:42:33 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:

>>The overlapping is important if you want to pretend that the
>>namespace-able resources are allowed to be specified per-process, when
>>really they are specified per-family.
>>
>>In this way, a process family is merely a grouping of processes with
>>like namespaces, and depending on which way they overlap you get the
>>same behaviour as when processes only have one resource different, and
>>therefore remove the overhead on fork().
>>
>>
>
>
>I missed this subthread originally.
>
>I think it is important that we can have containers in containers

>if at all possible. This means large software collections can count
>on them being present.
>
>

Right. Well, my concept was that that "in"-ness is just a relationship between two process families, so treat it relationally and not heirarchically. So, to the kernel, they've all got global unique IDs, but to the actual userspace within those families, they might see something different. And the model still supports containers in containers.

>As for having some items inside a namespace show up in both
>a parent and a child namespace I think the case is less clearly
>defined. If possible that is something we want to avoid as it
>complicates the implementation.
>
>For pids I will be surprised if we can avoid it.
>
>For most other namespaces I think we can, and it is a good thing
>to avoid.
>
>

Well, let's cross those bridges when we come to them. I agree it should only be implemented if required for individual numbers. PIDs and process family IDs seem logical to me, at this point anyway.

Sam.
