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.
