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