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