Subject: Re: Checkpoint/restart (was Re: [PATCH 0/4] - v2 - Object creation with a specified id)
Posted by Oren Laadan on Thu, 24 Apr 2008 01:19:51 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:
>> If the current interface is insufficient, we should first expand it in
>> such a way that it can be used for checkpoint. That certainly won't
>> work in all cases. fork(), for instance, doesn't take any arguments and
>> is going to be awfully hard to expand. :)
>>
>> I'd love to hear some of your insights about how things like the current
>> iptables interfaces are insufficient for checkpoint/restart.
>
> iptables is a bad example. Luckily for checkpointing - it always had an interface
> "load full state", "dump full state".
>
> But even iptables are not working in current form for checkpointing - they can't
> save/restore state of conntracks. Do you think netdev@/netfilter@ guys will be happy
> to have APIs allowing to set conntracks and have all the pain related
> to API stability - cause conntrack state changed a couple of times during last 2 years.
>
> Consider more intimate kernel states like:
> a. task statistics
> b. task start time
> c. load average
> d. skb state and it's data.
> e. mount tree.
>
> If you think over, e.g. (b) is a bad thing. It was used to be accounted in jiffies, then in timespec.
> (a) is another example of dataset which we can't predict. task statistics change over a time.
> Why bother with such intimate data in user-space at all?
> Why the hell user-space should know about it and be ABLE to modify it?

Agreed.

> Why do we need to export ability to set IDs for some objects which none of the
> operating systems do and then to have a burden to support it for application compatibility
> the rest of our lifes? Do you really believe none of the applications except for checkpointing
> will be using it?
>
> My personal vision is that:
> 1. user space must initialize checkpointing/restore state via some system call,
>   supply file descriptor from where data can be read/written to.
> 2. must call the syscall asking kernel to restore/save different subsytems one by one.
> 3. finalize cpt/restore state via the syscall
> But user-space MUST NOT bother about data content. At least not about the data supplied by
the kernel.

> It can add additional sections if needed, e.g. about iptables state.

I mostly agree with the vision that checkpoint/restart is probably best
implemented as a black box:

* First, much of the work required to restore the state of a process
as well as the state of its resources, requires kernel interfaces that
are lower than the ones available to user-space. Working in user-space
will require that we design new complex interfaces for this purpose only.

* Second, much of the state that needs to be saved was not, is not, and
should probably never be exported to user-space (e.g. interval socket
buffers, t->did_exec and many more). It is only accessible to kernel
code, so an in-kernel module (for checkpoint/restart) makes sense. It is
that sort of internals that may (and will) change as the kernel evolves
- precisely because it is not visible to user-space and not bound to it.

That said, we should still attempt to reuse existing kernel interfaces
and mechanisms as much as possible to save - and restore - the state of
processes, and prefer that over handcrafting special code. This is
especially true for restart: in checkpoint one has to _capture_ the
state by probing it in a passive manner; in contrast, in restart one
has to actively _construct_ new resources and ensure that their state
matches the saved state.

For instance, it is possible to create the process tree in a container
during restart from user-space reusing clone() (I'd argue that it's
even better to do so from user-space). Likewise, it is possible to redo
an open file by opening the file then using dup2() syscall to adjust
the target fd if necessary. The two differ in that the latter allows
to adjust the (so called) resources identifier to the desired value
(because it is privately visible), while the former does not - it gives
a globally visible identifier. And this is precisely why this thread
had started: how to determine the resource identifier when requesting
to allocate a resource (in this example, the pid).

>
> Having all this functionality in a signle syscall we specifically CLAIM a black box,
> and that no one can use this interfaces for something different from checkpoint/restore.

True, except for what can be done (and is easier to actually that way)
in user space; the most obvious example being clone() and setsid() -
which are a pain to adjust after the fact. In particular, everything
that is private in the kernel now (un-exported) should remain that way,
unless there is an (other) compelling reason to expose it.
(see http://www.ncl.cs.columbia.edu/publications/usenix2007_fordist.pdf)

Regardless of this black-box approach, we need a method to tell the

kernel code that we reuse, that we want a certain resource to have a
certain value. Whether or not this method is exported to user space
depends on whether or not that particular resource is constructed from
user space or not.

For example, Zap uses a special per-task (task_struct) field that if
set designates the next resource identifier expected to be used. It
is set before restore of pid's (clone), IPC (all sorts) and pseudo
terminals. Of these, only clone() is called from user-space, so the
interface allows only that one to be set from user-space.

Oren.

>
> So I think we have to know what other maintainers think before we can go.
>
>
>>> These next ids are suitable, well, only for ids which is very, very small
>>> part of kernel state needed to restore group of processes.
>> I couldn't agree more. This id setting mechanism would only be useful
>> for a small subset of the things we need during a restart.
>>
>> -- Dave
>>
>> ____
>> Containers mailing list
>> Containers@lists.linux-foundation.org
>> https://lists.linux-foundation.org/mailman/listinfo/containers
>>
> ____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers