Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by Pavel Emelianov on Fri, 25 May 2007 07:15:51 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Pavel Emelianov <xemul@openvz.org> writes:
>
>> Eric W. Biederman wrote:
>>> Pavel Emelianov <xemul@openvz.org> writes:
>>>
>>>> That's how OpenVZ sees the pid namespaces.
>>>>
>>>> The main idea is that kernel keeps operating with tasks pid
>>>> as it did before, but each task obtains one more pid for each
>>>> pid type - the virtual pid. When putting the pid to user or
>>>> getting the pid from it kernel operates with the virtual ones.
>>> Just a quick reaction.
>>>
>>> - I would very much like to see a minimum of 3 levels of pids,
>> Why not 4? From my part, I would like to know, why such nesting
>> is important. We have plain IPC namespaces and nobody cares.
>> We will have isolated network namespaces, why pids are exception?
>
> 4+ is fine, and something we will probably care about someday.
> 3 seems to be the minimum necessary to get people thinking about
> adding more so we don't have arbitrary special cases, especially
> in the user interface. At 3 the things are simple enough we don't
> have to allocate additional data structures etc.
>
> If we don't need nesting we don't even need 2 levels, and we
> can remove the global pid. But we have had that conversation
> and especially for the current OpenVZ usage we need nesting.

We need nesting but 2 levels is more than enough. Yet again -
we have 2 level IPC namespace, 2 level network namespace etc.

Generic structures are not always needed. Say, why don't we
have N-level page tables in kernel? Why not make them generic?
What if some ia128 architecture will require 7-level tables!?

> Having more then two layers means we are prepared to use pid namespaces more
> generally. It really isn't that much harder.

It is not, but do we need to spend so much time on solving
not relevant problems?

>>> being supported. Otherwise it is easy to overlook some of the
>>> cases that are required to properly support nesting, which long

>>> terms seems important.
>>>
>>> - Semantically fork is easier then unshare. Unshare can mean
>> This is not. When you fork, the kid shares the session and the
>> group with its parent, but moving this pids to new ns is bad - the
>> parent will happen to be half-moved. Thus you need to break the
>> session and the group in fork(), but this is extra complexity.
>
> Nope. You will just need to have the child call setsid() if
> you don't want to share the session and the group.

Of course, but setsid() must be done *before* creating a new
namespace, Otherwise you will have a half-inserted into new
namespace task. This sounds awful.

> You can perfectly well share the sid and group with the parent,
> because internal to the kernel pids aren't numeric, they are struct
> pid pointers.
>
> There is the question of do you use foreign pid handling to display
> the session and the group, or do you allocate pids for the session
> and the group in the new pid namespace. At this point foreign pid
> handling looks sufficient.
>
>>> a lot of things, and it is easy to pick a meaning that has weird
>>> side effects. Your implementation has a serious problem in that you
>>> change the value of getpid() at runtime. Glibc does not know how to
>>> cope with the value of getpid() changing.
>> This pid changing happens only once per task lifetime.
>
> Unshare isn't once per task lifetime, unless you added some extra
> constraints.

It is once. You create a new namespace and that's all.

> Though I haven't
>> seen any problems with glibc for many years running OpenVZ and I think,
>> that if glibc will want to cache this getpid() value we can teach it to
>> uncache this value in case someone called unshare() with CLONE_NEWPIDS.
>
> glibc very much caches the results of getpid().

Can you prove it? We have run OpenVZ for many years and with many
userspace configurations and we haven't seen the problems with
glibc ever.

> If you want to teach glibc not to cache getpid() fee free. The only
> way I know to get glibc to invalidates it's pid cache is to call fork.

>
> Eric
>