Subject: Re: [PATCH 0/13] Pid namespaces (OpenVZ view)
Posted by ebiederm on Tue, 29 May 2007 13:07:13 GMT
View Forum Message <> Reply to Message

Hmm. I seem to have forgotten to send this one.

Pavel Emelianov <xemul@openvz.org> writes:

> Eric W. Biederman wrote:

> 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!?

PID namespaces unlike the other namespaces are fundamentally nested.
Which is an unfortunate pain. But if you want to allow nesting
of containers of different types such as system containers and
application containers you need nested PID namespaces.

>> 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?

It is relevant to some of us. Therefore it is a relevant problem.

>>>> - 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.

We can experience weird interactions, but not really worse then the
sending a signal from outside the namespace. So we may want to
map the pids of the session and the pgrp into the new namespace but
functionally it's not really a big deal, and we can call setsid
after the fork.

>>>>   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.

What prevents you from calling unshare multiple times?

>>> 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.

Yes. I did a migration prototype in user space. Migrated a process
to a new pid, but getpid returned the pid before migration. So I
investigated why, including reading the glibc code. glibc cache the
pid value. Once the value is cached only a fork invalidates the
cache.

From: nptl/sysdeps/unix/sysv/linux/getpid.c

```
pid_t
__getpid (void)
{
  pid_t result = THREAD_GETMEM (THREAD_SELF, pid);
  if (__builtin_expect (result <= 0, 0))
    result = really_getpid (result);
  return result;
}
```

THREAD_GETMEM is a memory read.
really_getpid is the syscall.

Eric