

---

Subject: Re: [RFC][PATCH 4/4] PID: use the target ID specified in procs  
Posted by [ebiederm](#) on Wed, 12 Mar 2008 19:58:56 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

"Nadia Derby" <Nadia.Derbey@bull.net> writes:

> A couple of weeks ago, a discussion has started after Pierre's proposal for  
> a new syscall to change an ipc id (see thread  
> <http://lkml.org/lkml/2008/1/29/209>).

>

>

> Oren's suggestion was to force an object's id during its creation, rather  
> than 1. create it, 2. change its id.

>

> So here is an implementation of what Oren has suggested.

>

> 2 new files are defined under /proc/self:

> . next\_ipcid --> next id to use for ipc object creation

> . next\_pids --> next upid nr(s) to use for next task to be forked

> (see patch #2 for more details).

>

> When one of these files (or both of them) is filled, a structure pointed to  
> by the calling task struct is filled with these ids.

>

> Then, when the object is created, the id(s) present in that structure are  
> used, instead of the default ones.

> A couple of weeks ago, a discussion has started after Pierre's proposal for  
> a new syscall to change an ipc id (see thread

> <http://lkml.org/lkml/2008/1/29/209>).

>

>

> Oren's suggestion was to force an object's id during its creation, rather  
> than 1. create it, 2. change its id.

>

> So here is an implementation of what Oren has suggested.

>

> 2 new files are defined under /proc/self:

> . next\_ipcid --> next id to use for ipc object creation

> . next\_pids --> next upid nr(s) to use for next task to be forked

> (see patch #2 for more details).

>

> When one of these files (or both of them) is filled, a structure pointed to  
> by the calling task struct is filled with these ids.

>

> Then, when the object is created, the id(s) present in that structure are  
> used, instead of the default ones.

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> Right the alloc\_pidmap() changes will probably be pretty much the same  
> no matter how we do set\_it(), so it's worth discussing. But I'm  
> particularly curious to see what opinions are on the sys\_setid().

A couple of comments. With respect to alloc\_pidmap we already have the necessary controls (a minimum and a maximum) in place for the allocation. So except for double checking that those controls are exported in /proc/sys we don't necessarily need to do anything, special.

Just play games with the minimum pid value before you fork.

Second at least to get the memory map correct we need additional kernel support.

Third to actually get the values out it appears we need additional kernel support as well. From my limited playing with these things at least parts of the code were easier to implement in the kernel. The hoops you have to go to restore a single process (without threads) are absolutely horrendous in user space.

So this patchset whether it is setid or setting the id at creation time seems to be jumping the gun. For some namespaces renames are valid and we can support them. For other namespaces setting the id is a big no-no, and possibly even controlling the id at creation time is a problem (at least in the general sense). Because if you can easily control the id you may be able to more easily exploit security holes.

I'm not at all inclined to make it easy for userspace to rename or set the id of a new resource unless it already makes sense in that namespace.

We need to limit anything in a checkpoint to user space visible state. For sockets this can get fairly abstract since on the wire state of a tcp socket is in some sense user visible. However it should not include things that user space can never see like socket hash values.

Partly what this set of patches demonstrates is that it is fairly straight forward to restore ids. Getting the little details of the proper maintainable user space interface correct is harder.

The goal with any userspace implementation is to that we can separate policy form mechanism. So what are the trade offs for various approaches.

At least for inspection at the checkpoint side it would be nice for debugging applications to easily get at all of the user space visible state. So there is an argument for making state that is only indirectly visible, visible for diagnostic and debugging purposes.

For saving the state to disk it appears we need to stop all of the processes in our container. Again something a debugging application of an entire container may want to do. Although we also want to stop the hardware queues for things like networking so we don't transmit or possibly receive new packets either.

We want a checkpoint/restart to be essentially atomic, with non of the tasks that we stop being able to prove that they ran while the checkpoint was being taken. Mostly this is an interprocess communication blackout, but there may be more to it than that.

We want checkpoint/restart if possible to be incremental. So we can perform actions like live migration efficiently if most of the data is not changing.

So there is an argument to perform the work piecemeal instead of in one big shot. Although if we can load data from wherever into the kernel data structures quickly it may not be a big deal.

We also want the transfer of state to be fast. Which tends to argue in the other direction. That we want a bulk operation that can save out everything and restore everything quickly.

We also want a design that we can implement incrementally. So that we can avoid supporting everything at first and if there is state that we should save that we can't (or similarly state that we should restore but we can't) the save/restore fails. Until that part is implemented.

Further we need to finish difficult things like sysfs support and proc support for simply running applications in containers.

So while I think it is good to be thinking and playing with these ideas now. I think having a more complete story and not pecking on the pieces right now is important.

My inclination is that create with a specified set of ids is the proper internal kernel API, so we don't have rework things later, because reworking things seems to be a lot more work. How we want to export this to user space is another matter.

One suggestion is to have /proc or a proc like filesystem that

allows us to read, create, and populate files to see all of the application state. Then in userspace it is possible that the transfer of the checkpoint would be as simple as rsync.

Ok back to my cave for a bit.

Eric

---

Containers mailing list  
Containers@lists.linux-foundation.org  
<https://lists.linux-foundation.org/mailman/listinfo/containers>

---