Subject: Re: [PATCH 2.6.24-rc8-mm1 09/15] (RFC) IPC: new kernel API to change an ID
Posted by serue on Wed, 06 Feb 2008 05:00:39 GMT
View Forum Message <> Reply to Message

Quoting Oren Laadan (orenl@cs.columbia.edu):
>
>
> Serge E. Hallyn wrote:
>> Quoting Oren Laadan (orenl@cs.columbia.edu):
>>> I strongly second Kirill on this matter.
>>>
>>> IMHO, we should _avoid_ as much as possible exposing internal kernel
>>> state to applications, unless a _real_ need for it is _clearly_
>>> demonstrated. The reasons for this are quite obvious.
>> Hmm, sure, but this sentence is designed to make us want to agree.  Yes,
>> we want to avoid exporting kernel internals, but generally that means
>> things like the precise layout of the task_struct.  What Pierre is doing
>> is in fact the opposite, exporting resource information in a kernel
>> version invariant way.
>
> LOL ... a bit of misunderstanding - let me put some order here:
>
> my response what with respect to the new interface that Pierre
> suggested, that is - to add a new IPC call to change an identifier
> after it has been allocated (and assigned). This is necessary for the
> restart because applications expect to see the same resource id's as
> they had at the time of the checkpoint.
>
> What you are referring to is the more recent part of the thread, where
> the topic became how data should be saved - in other words, the format
> of the checkpoint data. This is entirely orthogonal to my argument.
>
> Now please re-read my email :)

Heh - by the end of my response I was pretty sure that was the case :)

> That said, I'd advocate for something in between a raw dump and a pure
> "parametric" representation of the data. Raw data tends to be, well,
> too raw, which makes the task of reading data from older version by
> newer kernels harder to maintain. On the other hand, it is impossible
> to abstract everything into kernel-independent format.

Well, that's probably getting a little pedantic, but true.

>> In fact, the very reason not to go the route you and Pavel are
>> advocating is that if we just dump task state to a file or filesystem
>> from the kernel in one shot, we'll be much more tempted to lay out data

>> in a way that exports and ends up depending on kernel internals.  So
>> we'll just want to read and write the task_struct verbatim.
>> So, there are two very different approaches we can start with.
>> Whichever one we follow, we want to avoid having kernel version
>> dependencies.  They both have their merits to be sure.
>
> You will never be able to avoid that completely, simply because new
> kernels will require saving more (or less) data per object, because
> of new (or dropped) features.

Sure.

> The best solution in this sense is to provide a filter (hopefully
> in user space, utility) that would convert a checkpoint image file
> from the old format to a newer format.

Naturally.

> And you keep a lot of compatibility code of the kernel, too.
>
>> But note that in either case we need to deal with a bunch of locking.
>> So getting back to Pierre's patchset, IIRC 1-8 are cleanups worth
>> doing no matter 1.  9-11 sound like they are contentuous until
>> we decide whether we want to go with a create_with_id() type approach
>> or a set_id().  12 is IMO a good locking cleanup regardless.  13 and
>> 15 are contentous until we decide whether we want userspace-controlled
>> checkpoint or a one-shot fs.  14 IMO is useful for both c/r approaches.
>> Is that pretty accurate?
>
> (context switch back to my original reply)
>
> I prefer not to add a new interface to IPC that will provide a new
> functionality that isn't needed, except for the checkpoint - because
> there is a better alternative to do the same task; this alternative
> is more suitable because (a) it can be applied incrementally, (b) it
> provides a consistent method to pre-select identifiers of all syscalls,
> (where is the current suggestion suggests one way for IPC and will
> suggest other hacks for other resources).
>
> (context switch back to the current reply)
>
> I definitely welcome a cleanup of the (insanely multiplexedd) IPC
> code. However I argue that the interface need not be extended.
>
>>> It isn't strictly necessary to export a new interface in order to
>>> support checkpoint/restart. **. Hence, I think that the speculation
>>> "we may need it in the future" is too abstract and isn't a good
>>> excuse to commit to a new, currently unneeded, interface.

>> OTOH it did succeed in starting some conversation :)
>>> Should the
>>> need arise in the future, it will be easy to design a new interface
>>> (also based on aggregated experience until then).
>> What aggregated experience?  We have to start somewhere...
>
> :)  well, assuming the selection of resource IDs is done as I suggested,
> we'll have the restart use it. If someone finds a good reason (other
> than checkpoint/restart) to pre-select/modify an identifier, it will
> be easy to _then_ add an interface. That (hypothetical) interface is
> likely to come out more clever after X months using checkpoint/restart.
>
>>> ** In fact, the suggested interface may prove problematic (as noted
>>> earlier in this thread): if you first create the resource with some
>>> arbitrary identifier and then modify the identifier (in our case,
>>> IPC id), then the restart procedure is bound to execute sequentially,
>>> because of lack of atomicity.
>> Hmm?  Lack of atomicity wrt what?  All the tasks being restarted were
>> checkpointed at the same time so there will be no conflict in the
>> requested IDs, so I don't know what you're referring to.
>
> Consider that we want to have an ultra-fast restart, so we let processes
> restart in parallel (as much as possible) in the same container. Task A
> wants to allocate IPC id 256, but the kernel allocates 32; before task A
> manages to change it to 256 (with the new interface), task B attempts to
> create an IPC id 32; the kernel provides, say, 1024, and task B fails to
> change it to 32 because it is still used by task A. So restart fails :(

Bah, it gets -EAGAIN and tries again.  I see the biggest plus of your
approach as being the consistent api.

> On the other hand, if a process first tells the kernel "I want 32" and
> then calls, for instance, semget(), then the IPC can atomically ensure
> that the process gets what it wanted.
>
>>> That said, I suggest the following method instead (this is the method
>>> we use in Zap to determine the desired resource identifier when a new
>>> resource is allocated; I recall that we had discussed it in the past,
>>> perhaps the mini-summit in september ?):
>>>
>>> 1) The process/thread tells the kernel that it wishes to pre-determine
>>> the resource identifier of a subsequent call (this can be done via a
>>> new syscall, or by writing to /proc/self/...).
>>>
>>> 2) Each system call that allocates a resource and assigns an identifier
>>> is modified to check this per-thread field first; if it is set then
>>> it will attempt to allocate that particular value (if already taken,
>>> return an error, eg. EBUSY). Otherwise it will proceed as it is today.

>> But I thought you were just advocating a one-shot filesystem approach
>> for c/r, so we wouldn't be creating the resources piecemeal?
>
> I wasn't. That was Pavel. While I think the idea is neat, I'm not
> convinced that it's practical and best way to go, however I need to
> further think about it.
>
> And as I said, I see this as a separate issue from the problem of
> create_with_id()/set_id issue().
>
>> The /proc/self approach is one way to go, it has been working for LSMs
>> this long.  I'd agree that it would be nice if we could have a
>> consistent interface to the create_with_id()/set_id() problem.  A first
>> shot addressing ipcs and pids would be a great start.
>>> (I left out some details - eg. the kernel will keep the desire value
>>> on a per-thread field, when it will be reset, whether we want to also
>>> tag the field with its type and so on, but the idea is now clear).
>>>
>>> The main two advantages are that first, we don't need to devise a new
>>> method for every syscall that allocates said resources (sigh... just
>> Agreed.
>>> think of clone() nightmare to add a new argument);
>> Yes, and then there will need to be the clone_with_pid() extension on
>> top of that.
>
> Exactly !  With the /proc/self/... approach there will not be a need
> for a clone_with_pid() extension in terms of user-visible interface;
> makes the clone-flags headache a bit more manageable :p

So you say this is how zap does it now?  Would it be pretty trivial to
make a small patch consisting of your base procpid code and the clone
plugin to let you clone with a particular pid, and post that?

thanks,
-serge

> Ah... ok, long one, hopefully clarifies the confusion. That said, I
> suggest that the debate regarding the format of the checkpoint data
> shall proceed on a new thread, since IMHO it's orthogonal.
>
> Oren.
>
>>> second, the change
>>> is incremental: first code the mechanism to set the field, then add
>>> support in the IPC subsystem, later in the DEVPTS, then in clone and
>>> so forth.
>>>
>>> Oren.

>>>
>>> Pierre Peiffer wrote:
>>>> Kirill Korotaev wrote:
>>>>> Why user space can need this API? for checkpointing only?
>>>> I would say "at least for checkpointing"... ;) May be someone else may
>>>> find an
>>>> interest about this for something else.
>>>> In fact, I'm sure that you have some interest in checkpointing; and
>>>> thus, you
>>>> have probably some ideas in mind; but whatever the solution you will
>>>> propose,
>>>> I'm pretty sure that I could say the same thing for your solution.
>>>> And what I finally think is: even if it's for "checkpointing only", if
>>>> many
>>>> people are interested by this, it may be sufficient to push this ?
>>>>> Then I would not consider it for inclusion until it is clear how to
>>>>> implement checkpointing.
>>>>> As for me personally - I'm against exporting such APIs, since they are
>>>>> not needed in real-life user space applications and maintaining it
>>>>> forever for compatibility doesn't worth it.
>>>> Maintaining these patches is not a big deal, really, but this is not the
>>>> main
>>>> point; the "need in real life" (1) is in fact the main one, and then,
>>>> the "is
>>>> this solution the best one ?" (2) the second one.
>>>> About (1), as said in my first mail, as the namespaces and containers
>>>> are being
>>>> integrated into the mainline kernel, checkpoint/restart is (or will be)
>>>> the next
>>>> need.
>>>> About (2), my solution propose to do that, as much as possible from
>>>> userspace,
>>>> to minimize the kernel impact. Of course, this is subject to discussion.
>>>> My
>>>> opinion is that doing a full checkpoint/restart from kernel space will
>>>> need lot
>>>> of new specific and intrusive code; I'm not sure that this will be
>>>> acceptable by
>>>> the community. But this is my opinion only. Discusion is opened.
>>>>> Also such APIs allow creation of non-GPL checkpointing in user-space,
>>>>> which can be of concern as well.
>>>> Honestly, I don't think this really a concern at all. I mean: I've never
>>>> seen
>>>> "this allows non-GPL binary and thus, this is bad" as an argument to
>>>> reject a
>>>> functionality, but I may be wrong, and thus, it can be discussed as
>>>> well.
>>>> I think the points (1) and (2) as stated above are the key ones.

>>>> Pierre
>>>>> Kirill
>>>>>
>>>>>
>>>>> Pierre Peiffer wrote:
>>>>>> Hi again,
>>>>>>
>>>>>>  Thinking more about this, I think I must clarify why I choose this
>>>>>> way.
>>>>>> In fact, the idea of these patches is to provide the missing user APIs
>>>>>> (or
>>>>>> extend the existing ones) that allow to set or update _all_ properties
>>>>>> of all
>>>>>> IPCs, as needed in the case of the checkpoint/restart of an
>>>>>> application (the
>>>>>> current user API does not allow to specify an ID for a created IPC,
>>>>>> for
>>>>>> example). And this, without changing the existing API of course.
>>>>>>
>>>>>>  And msgget(), semget() and shmget() does not have any parameter we
>>>>>> can use to
>>>>>> specify an ID.
>>>>>>  That's why I've decided to not change these routines and add a new
>>>>>> control
>>>>>> command, IP_SETID, with which we can can change the ID of an IPC.
>>>>>> (that looks to
>>>>>> me more straightforward and logical)
>>>>>>
>>>>>>  Now, this patch is, in fact, only a preparation for the patch 10/15
>>>>>> which
>>>>>> really complete the user API by adding this IPC_SETID command.
>>>>>>
>>>>>> (... continuing below ...)
>>>>>>
>>>>>> Alexey Dobriyan wrote:
>>>>>>> On Tue, Jan 29, 2008 at 05:02:38PM +0100, pierre.peiffer@bull.net
>>>>>>> wrote:
>>>>>>>> This patch provides three new API to change the ID of an existing
>>>>>>>> System V IPCs.
>>>>>>>>
>>>>>>>> These APIs are:
>>>>>>>>  long msg_chid(struct ipc_namespace *ns, int id, int newid);
>>>>>>>>  long sem_chid(struct ipc_namespace *ns, int id, int newid);
>>>>>>>>  long shm_chid(struct ipc_namespace *ns, int id, int newid);
>>>>>>>>
>>>>>>>> They return 0 or an error code in case of failure.
>>>>>>>>
>>>>>>>> They may be useful for setting a specific ID for an IPC when

>>>>>>>> preparing
>>>>>>>> a restart operation.
>>>>>>>>
>>>>>>>> To be successful, the following rules must be respected:
>>>>>>>> - the IPC exists (of course...)
>>>>>>>> - the new ID must satisfy the ID computation rule.
>>>>>>>> - the entry in the idr corresponding to the new ID must be free.
>>>>>>>>  ipc/util.c          |   48
>>>>>>>> ++++++++++++++++++++++++++++++++++++++++++++++++
>>>>>>>>  ipc/util.h          |    1 +
>>>>>>>>  8 files changed, 197 insertions(+)
>>>>>>> For the record, OpenVZ uses "create with predefined ID" method which
>>>>>>> leads to less code. For example, change at the end is all we want
>>>>>>> from
>>>>>>> ipc/util.c .
>>>>>> And in fact, you do that from kernel space, you don't have the
>>>>>> constraint to fit
>>>>>> the existing user API.
>>>>>> Again, this patch, even if it presents a new kernel API, is in fact a
>>>>>> preparation for the next patch which introduces a new user API.
>>>>>>
>>>>>> Do you think that this could fit your need ?
>>>>>>
>>> _______________________________________________
>>> 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