Subject: Re: [RFC][PATCH 0/4] Object creation with a specified id
Posted by Oren Laadan on Sun, 16 Mar 2008 19:08:44 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Oren Laadan (orenl@cs.columbia.edu):
>>
>> Nadia Derbey wrote:
>>> Oren Laadan wrote:
>>>>
>>>> Nadia Derbey wrote:
>>>>
>>>>> Oren Laadan wrote:
>>>>>
>>>>>>
>>>>>> Nadia.Derbey@bull.net wrote:
>>>>>>
>>>>>>> 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).
>>>>>>
>>>>>>
>>>>>> Generally looks good. One meta-comment, though:
>>>>>>
>>>>>> I wonder why you use separate files for separate resources,
>>>>>
>>>>> That would be needed in a situation wheere we don't care about next,
>>>>> say, ipc id to be created but we need a predefined pid. But I must
>>>>> admit I don't see any pratical application to it.
>>>>
>>>> exactly; why set the next-ipc value so far in advance ? I think it's
>>>> better (and less confusing) if we require that setting the next-id value
>>>> be done right before the respective syscall.
>>> Ok, but this "requirement" should be widely agreed upon ;-)
>> A discussion on the overall checkpoint/restart policy is certainly due

>> (and increasingly noted recently).
>>
>>> What I mean here is that the solution with 1 file per "object type" can
>>> easily be extended imho:
>> I'm aiming at simplicity and minimal (but not restrictive) API for user
>> space. I argue that we never really need more than one predetermined value
>> at a time (eg see below), and the cost of setting such value is so small
>> that there is no real benefit in setting more than one at a time (either
>> via multiple files or via an array of values). If in fact you wanted more
>> than one type at a time, you could still make it happen with a single
>> file without adding many user-visible files in /proc/<pid>.
>>
>> So far, I can't think of any such identifier that we'd like to pre-set
>> that does not fit into a "long" type;
>
> As Nadia has mentioned, if we have checkpointed a container which has
> another pid namespace underneath itself, then we will need to restart
> some tasks with two predetermined pids. So we'll need two (or more)
> longs for the tasks in deeper namespaces.

I see. So more than a single "long" type is probably needed. I'd still
prefer that the "scope" of a preset identifier through "next_id" should
be the subsequent syscall; so if you need multiple values for the next
syscall you use it, but you don't support leftovers for the next syscall
to use. The typing system can be something like "long VAL" and then for
array "long* VAL VAL VAL ...", for instance.

>
>> simply because the kernel does not
>> use such identifiers in the first place (pid, ipc, pty#, vc# .. etc). To
>> be on the safe side, we can require that the format be "long VAL", just
>> in case (and later you could have other formats).
>>
>> The only exception, perhaps, is if a TCP connection is rebuilt with a,
>> say, connect() syscall, and some information needs to be "predetermined"
>> so we'll need to extend the format. That can be done with another type
>> eg. "tcp ....." or a separate file (per your view), _then_, not now.
>> (As a side note, I don't suggest that this is how TCP will be restored).
>>
>> In any event, the bottom line is that a single file, with a single
>> value at a time (possibly annotated with a type), is the simplest, and
>> isn't restrictive, for our purposes. Looking one step ahead, simplicity
>> and minimal commitment to user space is important in trying to push this
>> to the mainline kernel...
>>
>>> I don't know how the restart is supposed to work, but we can imagine
>>> feeding all these files with all the object ids just before restart and
>> Building on my own experience with zap I envision the restart operation

>> of a given task occurring in the context of that task.
>
> Could be, but not necessarily the case. Eric has mentioned using elf
> files for restart, and that's one way to go, but whether one central

I'm not familiar with the details of this.

> restart task sets up all the children or the children set themselves up
> is yet another design point we haven't decided. I would think that
> with a centralized restart it would be easier to assure for instance
> that shared anon pages would be properly set up and shared, but since
> you advocate each-task-starts-itself I trust zap must handle that.

The main reason I think a task should setup itself, is because most of
the setup requires that new resources be allocated, and the kernel is
already centered around this approach that a task allocates for itself,
not for another task. For instance, if you need to restore a VMA, you
simply call mmap(), a new file, you call open() etc.

Shared anon pages are one example of shared resources that may be used
by multiple processes. Zap's approach is to have the "first" user (in
the sense of the first time the resource is seen during checkpoint) do
the actual restore, and place it in a global table, and then subsequent
tasks will find it in the table and "map" it into their view.

Decentralizing also allow multiple tasks to restart concurrently.

Are we ready to start concrete discussion on the architecture for the
checkpoint/restart ? (and if so .. time to change the subject line).

>
>> (I assume this is
>> how restart will work). Therefore, it makes much sense that before every
>> syscall that requires a pre-determined resource identifier (eg. clone,
>> ipc, pty allocation), the task will place the desired value in "next_id"
>> (and that will only be meaningful during restart) and invoke the said
>> syscall. Voila.
>>
>> Note that the restart will "rebuild" the container's state (and the task
>> state) as it reads in the data from some source. It is likely that not
>> all data will be available when the first said syscall is about to be
>> invoked, so you may not be able to feed everything ahead of time.
>>
>>
>>> let the process pick up the objects ids as it needs them.
>>> Of course, this would require to enhance the files formats, as well as
>>> the way things are stored in the task_struct.
>>>

>>> Hope what I'm saying is not too stupid ;-) ?
>>>
>>> Regards,
>>> Nadia
>>>
>>>>>> and why you'd
>>>>>> want to write multiple identifiers in one go;
>>>>>
>>>>> I used multiple identifiers only for the pid values: this is because
>>>>> when a new pid value is allocated for a process that belongs to
>>>>> nested namespaces, the lower level upid nr values are allocated in a
>>>>> single shot. (see alloc_pid()).
>>>>>
>>>>>> it seems to complicate the
>>>>>> code and interface with minimal gain.
>>>>>> In practice, a process will only do either one or the other, so a
>>>>>> single
>>>>>> file is enough (e.g. "next_id").
>>>>>> Also, writing a single value at a time followed by the syscall is
>>>>>> enough;
>>>>>> it's definitely not a performance issue to have multiple calls.
>>>>>> We assume the user/caller knows what she's doing, so no need to
>>>>>> classify
>>>>>> the identifier (that is, tell the kernel it's a pid, or an ipc id)
>>>>>> ahead
>>>>>> of time. The caller simply writes a value and then calls the relevant
>>>>>> syscall, or otherwise the results may not be what she expected...
>>>>>> If such context is expected to be required (although I don't see any at
>>>>>> the moment), we can require that the user write "TYPE VALUE" pair to
>>>>>> the "next_id" file.
>>>>>
>>>>> That's exactly what I wanted to avoid by creating 1 file per object.
>>>>> Now, it's true that in a restart context where I guess that things
>>>>> will be done synchronously, we could have a single next_id file.
>>>>>
>>>>>>> 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.
>>>>>>>
>>>>>>> The patches are against 2.6.25-rc3-mm1, in the following order:
>>>>>>>
>>>>>>> [PATCH 1/4] adds the procfs facility for next ipc to be created.
>>>>>>> [PATCH 2/4] adds the procfs facility for next task to be forked.
>>>>>>> [PATCH 3/4] makes use of the specified id (if any) to allocate the

>>>>>>> new IPC
>>>>>>>   object (changes the ipc_addid() path).
>>>>>>> [PATCH 4/4] uses the specified id(s) (if any) to set the upid nr(s)
>>>>>>> for a newly
>>>>>>>   allocated process (changes the
>>>>>>> alloc_pid()/alloc_pidmap() paths).
>>>>>>>
>>>>>>> Any comment and/or suggestions are welcome.
>>>>>>>
>>>>>>> Cc-ing Pavel and Sukadev, since they are the pid namespace authors.
>>>>>>>
>>>>>>> Regards,
>>>>>>> Nadia
>>>>>>>
>>>>>>> --
>>>>>>>
>>>>>>> --
>>>>>>
>>>>>>
>>>>>>
>>>>>
>>>>> Regards,
>>>>> Nadia
>>>>
>>>>
>>>
>> ____
>> 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