Subject: Namespaces exhausted CLONE_XXX bits problem
Posted by Pavel Emelianov on Mon, 14 Jan 2008 13:45:08 GMT
View Forum Message <> Reply to Message

Hi, guys!

I started looking at PTYs/TTYs/Console to make the appropriate
namespace and suddenly remembered that we have already
exhausted all the CLONE_ bits in 32-bit mask.

So, I recalled the discussions we had and saw the following
proposals of how to track this problem (with their disadvantages):

1. make the clone2 system call with 64-bit mask
   - this is a new system call
2. re-use CLONE_STOPPED
   - this will give us only one bit
3. merge existing bits into one
   - we lose the ability to create them separately
4. implement a sys_unshare_ns system call with 64bit/arbitrary mask
   - this is anew system call
   - this will bring some dissymmetry between namespaces
5. use sys_indirect
   - this one is not in even -mm tree yet and it's questionable
     whether it will be at all

I have one more suggestion:

6. re-use bits, that don't make sense in sys_unshare (e.g.
   CLONE_STOPPED, CLONE_PARENT_SETTID, CLONE_VFORK etc)
   This will give us ~16 new bits, but this will look not very nice.

What do you think about all of this?

Thanks,
Pavel

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

Subject: Re: Namespaces exhausted CLONE_XXX bits problem
Posted by Cedric Le Goater on Mon, 14 Jan 2008 14:44:25 GMT
View Forum Message <> Reply to Message

Hello Pavel !

Pavel Emelyanov wrote:
> Hi, guys!
>
> I started looking at PTYs/TTYs/Console to make the appropriate
> namespace and suddenly remembered that we have already
> exhausted all the CLONE_ bits in 32-bit mask.

yes nearly. 1 left with the mq_namespace i'm going to send.

> So, I recalled the discussions we had and saw the following
> proposals of how to track this problem (with their disadvantages):
>
> 1. make the clone2 system call with 64-bit mask
> - this is a new system call

sys_clone2 is used on ia64 ... so we would need another name.

clone_ns() would be nice but it's too specific to namespaces unless
we agree that we need a new syscall specific to namespaces.

clone_new or clone_large ?

> 2. re-use CLONE_STOPPED
> - this will give us only one bit

not enough.

> 3. merge existing bits into one
> - we lose the ability to create them separately

it would be useful to have such a flag though, something like CLONE_ALLN,
because it's the one everyone is going to use.

what i've been looking at in December is 1. and 3. : a new general purpose
clone syscall with extend flags. The all-in-on flag is not an issue but it
would be nice to keep the last clone flag for this purpose.

Now, if we use 64bits, we have a few issue/cleanups to solve. First, in
kernel land, the clone_flags are passed down to the security modules

 security_task_create()

so we'll have to change to kernel api. I don't remember anything else
blocking.

In user land, we need to choose a prototype supporting also 32bits arches.
so it could be :

long sys_clone_new(struct clone_new_args)

or

long sys_clone_new(... unsigned long flags_high, unsigned long flag_low ...)

Second option might be an issue because clone already has 6 arguments.
right ?

> 4. implement a sys_unshare_ns system call with 64bit/arbitrary mask
> - this is anew system call

I think that a new clone deserves a new unshare.

> - this will bring some dissymmetry between namespaces

what do you mean ?

> 5. use sys_indirect
> - this one is not in even -mm tree yet and it's questionable
> whether it will be at all

I don't know much about that one.

C.

> I have one more suggestion:
>
> 6. re-use bits, that don't make sense in sys_unshare (e.g.
> CLONE_STOPPED, CLONE_PARENT_SETTID, CLONE_VFORK etc)
> This will give us ~16 new bits, but this will look not very nice.
>
> What do you think about all of this?
>
> Thanks,
> Pavel
> _____
> 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

Subject: Re: Namespaces exhausted CLONE_XXX bits problem
Posted by Pavel Emelianov on Mon, 14 Jan 2008 14:50:44 GMT
View Forum Message <> Reply to Message

Cedric Le Goater wrote:
> Hello Pavel !
>
> Pavel Emelyanov wrote:
>> Hi, guys!
>>
>> I started looking at PTYs/TTYs/Console to make the appropriate
>> namespace and suddenly remembered that we have already
>> exhausted all the CLONE_ bits in 32-bit mask.
>
> yes nearly. 1 left with the mq_namespace i'm going to send.

Yup. That's why I think that we should first solve this
issue and then send more namespaces.

>> So, I recalled the discussions we had and saw the following
>> proposals of how to track this problem (with their disadvantages):
>>
>> 1. make the clone2 system call with 64-bit mask
>>   - this is a new system call
>
> sys_clone2 is used on ia64 ... so we would need another name.
>
> clone_ns() would be nice but it's too specific to namespaces unless
> we agree that we need a new syscall specific to namespaces.
>
> clone_new or clone_large ?

clone3 :) Just kidding. _If_ implement new system calls then I'd
better like cloe_ns and unshare_nr pair.

>> 2. re-use CLONE_STOPPED
>>   - this will give us only one bit
>
> not enough.

Yup :)

>> 3. merge existing bits into one
>>   - we lose the ability to create them separately
>
> it would be useful to have such a flag though, something like CLONE_ALLN,
> because it's the one everyone is going to use.
>
> what i've been looking at in December is 1. and 3. : a new general purpose

> clone syscall with extend flags. The all-in-on flag is not an issue but it
> would be nice to keep the last clone flag for this purpose.
>
> Now, if we use 64bits, we have a few issue/cleanups to solve. First, in
> kernel land, the clone_flags are passed down to the security modules
>
> security_task_create()
>
> so we'll have to change to kernel api. I don't remember anything else
> blocking.
>
> In user land, we need to choose a prototype supporting also 32bits arches.
> so it could be :
>
> long sys_clone_new(struct clone_new_args)
>
> or
>
> long sys_clone_new(... unsigned long flags_high, unsigned long flag_low ...)
>
> Second option might be an issue because clone already has 6 arguments.
> right ?

Yes.

>
>> 4. implement a sys_unshare_ns system call with 64bit/arbitrary mask
>>   - this is anew system call
>
> I think that a new clone deserves a new unshare.
>
>>   - this will bring some dissymmetry between namespaces
>
> what do you mean ?

I mean, that soe namespaces will be unshare-only, but some
clone-and-unshare.

>> 5. use sys_indirect
>>   - this one is not in even -mm tree yet and it's questionable
>>   whether it will be at all
>
> I don't know much about that one.
>
> C.

So you seem to prefer a "new system call" approach, right?

>> I have one more suggestion:
>>
>> 6. re-use bits, that don't make sense in sys_unshare (e.g.
>>   CLONE_STOPPED, CLONE_PARENT_SETTID, CLONE_VFORK etc)
>>   This will give us ~16 new bits, but this will look not very nice.
>>
>> What do you think about all of this?
>>
>> Thanks,
>> Pavel
>> _____
>> 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

---

Subject: Re: Namespaces exhausted CLONE_XXX bits problem
Posted by Cedric Le Goater on Mon, 14 Jan 2008 15:20:19 GMT
View Forum Message <> Reply to Message

>>> I started looking at PTYs/TTYs/Console to make the appropriate
>>> namespace and suddenly remembered that we have already
>>> exhausted all the CLONE_ bits in 32-bit mask.
>> yes nearly. 1 left with the mq_namespace i'm going to send.
>
> Yup. That's why I think that we should first solve this
> issue and then send more namespaces.

OK.

>>> So, I recalled the discussions we had and saw the following
>>> proposals of how to track this problem (with their disadvantages):
>>>
>>> 1. make the clone2 system call with 64-bit mask
>>>   - this is a new system call
>> sys_clone2 is used on ia64 ... so we would need another name.
>>
>> clone_ns() would be nice but it's too specific to namespaces unless
>> we agree that we need a new syscall specific to namespaces.
>>

>> clone_new or clone_large ?
>
> clone3 :) Just kidding. _If_ implement new system calls then I'd
> better like cloe_ns and unshare_nr pair.

We will find a name.

>>> 2. re-use CLONE_STOPPED
>>> - this will give us only one bit
>> not enough.
>
> Yup :)
>
>>> 3. merge existing bits into one
>>> - we lose the ability to create them separately
>> it would be useful to have such a flag though, something like CLONE_ALLN,
>> because it's the one everyone is going to use.
>>
>> what i've been looking at in December is 1. and 3. : a new general purpose
>> clone syscall with extend flags. The all-in-on flag is not an issue but it
>> would be nice to keep the last clone flag for this purpose.
>>
>> Now, if we use 64bits, we have a few issue/cleanups to solve. First, in
>> kernel land, the clone_flags are passed down to the security modules
>>
>> security_task_create()
>>
>> so we'll have to change to kernel api. I don't remember anything else
>> blocking.
>>
>> In user land, we need to choose a prototype supporting also 32bits arches.
>> so it could be :
>>
>> long sys_clone_new(struct clone_new_args)
>>
>> or
>>
>> long sys_clone_new(... unsigned long flags_high, unsigned long flag_low ...)
>>
>> Second option might be an issue because clone already has 6 arguments.
>> right ?
>
> Yes.
>
>>> 4. implement a sys_unshare_ns system call with 64bit/arbitrary mask
>>> - this is anew system call
>> I think that a new clone deserves a new unshare.
>>

>>>   - this will bring some dissymmetry between namespaces
>> what do you mean ?
>
> I mean, that soe namespaces will be unshare-only, but some
> clone-and-unshare.

OK. we still have that already. pid namespace for instance.

>>> 5. use sys_indirect
>>>   - this one is not in even -mm tree yet and it's questionable
>>>   whether it will be at all
>> I don't know much about that one.
>>
>> C.
>
> So you seem to prefer a "new system call" approach, right?

yes.

to be more precise :

 long sys_clone_something(struct clone_something_args args)

and

 long sys_unshare_something(struct unshare_something_args args)

The arg passing will be slower bc of the copy_from_user() but we will
still have the sys_clone syscall for the fast path.

C.

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

---

Subject: Re: Namespaces exhausted CLONE_XXX bits problem
Posted by serue on Mon, 14 Jan 2008 16:32:46 GMT
View Forum Message <> Reply to Message

Quoting Cedric Le Goater (clg@fr.ibm.com):
> to be more precise :
>
> long sys_clone_something(struct clone_something_args args)
>
> and
>

> long sys_unshare_something(struct unshare_something_args args)
>
> The arg passing will be slower bc of the copy_from_user() but we will
> still have the sys_clone syscall for the fast path.
>
> C.

I'm fine with the direction you're going, but just as one more option,
we could follow more of the selinux/lsm approach of first requesting
clone/unshare options, then doing the actual clone/unshare. So
something like

 sys_clone_request(extended_64bit_clone_flags)
 sys_clone(usual args)

or

 echo pid,mqueue,user,ipc,uts,net > /proc/self/clone_unshare
 clone()

-serge

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

---

Subject: Re: Namespaces exhausted CLONE_XXX bits problem
Posted by Pavel Emelianov on Mon, 14 Jan 2008 16:52:21 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Cedric Le Goater (clg@fr.ibm.com):
>> to be more precise :
>>
>> long sys_clone_something(struct clone_something_args args)
>>
>> and
>>
>> long sys_unshare_something(struct unshare_something_args args)
>>
>> The arg passing will be slower bc of the copy_from_user() but we will
>> still have the sys_clone syscall for the fast path.
>>
>> C.
>
> I'm fine with the direction you're going, but just as one more option,
> we could follow more of the selinux/lsm approach of first requesting

> clone/unshare options, then doing the actual clone/unshare. So
> something like
>
> sys_clone_request(extended_64bit_clone_flags)

What if we someday hit the 64-bit limit? :)

> sys_clone(usual args)
>
> or
>
> echo pid,mqueue,user,ipc,uts,net > /proc/self/clone_unshare
> clone()

Well, this is how sys_indirect() was intended to work. Nobody
liked it, so I'm afraid this will also not be accepted.

> -serge
>

Thanks,
Pavel

Subject: Re: Namespaces exhausted CLONE_XXX bits problem
Posted by serue on Mon, 14 Jan 2008 18:07:48 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> Serge E. Hallyn wrote:
> > Quoting Cedric Le Goater (clg@fr.ibm.com):
> >> to be more precise :
> >>
> >> long sys_clone_something(struct clone_something_args args)
> >>
> >> and
> >>
> >> long sys_unshare_something(struct unshare_something_args args)
> >>
> >> The arg passing will be slower bc of the copy_from_user() but we will
> >> still have the sys_clone syscall for the fast path.
> >>
> >> C.
> >

> > I'm fine with the direction you're going, but just as one more option,
> > we could follow more of the selinux/lsm approach of first requesting
> > clone/unshare options, then doing the actual clone/unshare. So
> > something like
> >
> > sys_clone_request(extended_64bit_clone_flags)
>
> What if we someday hit the 64-bit limit? :)
>
> > sys_clone(usual args)
> >
> > or
> >
> > echo pid,mqueue,user,ipc,uts,net > /proc/self/clone_unshare
> > clone()
>
> Well, this is how sys_indirect() was intended to work. Nobody
> liked it, so I'm afraid this will also not be accepted.

I would have thought sys_indirect would be disliked because
it looks like an ioctl type multiplexor. Whereas sys_clone_request()
or /proc/self/clone_unshare simply sets arguments in advance, the
way /proc/self/attr/current does.

-serge

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

---

Subject: Re: Namespaces exhausted CLONE_XXX bits problem
Posted by Oren Laadan on Mon, 14 Jan 2008 21:36:13 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelyanov (xemul@openvz.org):
>> Serge E. Hallyn wrote:
>>> Quoting Cedric Le Goater (clg@fr.ibm.com):
>>>> to be more precise :
>>>>
>>>> long sys_clone_something(struct clone_something_args args)
>>>>
>>>> and
>>>>
>>>> long sys_unshare_something(struct unshare_something_args args)
>>>>
>>>> The arg passing will be slower bc of the copy_from_user() but we will

>>>> still have the sys_clone syscall for the fast path.
>>>>
>>>> C.
>>> I'm fine with the direction you're going, but just as one more option,
>>> we could follow more of the selinux/lsm approach of first requesting
>>> clone/unshare options, then doing the actual clone/unshare. So
>>> something like
>>>
>>> sys_clone_request(extended_64bit_clone_flags)
>> What if we someday hit the 64-bit limit? :)
>>
>>> sys_clone(usual args)

One (security ?) problem with a two stage approach is that the operation
may not be completed in an atomic manner; e.g. if there are two threads
doing the first call before any of them gets to the second call. Or at
least ensure that such races cannot occur by design. (In contrast, with
sys_indirect() everything is atomic).

Also, in a two-step approach, using /proc as opposed to a specialized
system call incurs higher overhead should ultra-fast clone()s are a
goal by itself.

I second the concern of running out of 64 bits of flags. In fact, the
problem with the flags is likely to be valid outside our context, and
general to the linux kernel soon. Should we not discuss it there too ?

>>>
>>> or
>>>
>>> echo pid,mqueue,user,ipc,uts,net > /proc/self/clone_unshare
>>> clone()
>> Well, this is how sys_indirect() was intended to work. Nobody
>> liked it, so I'm afraid this will also not be accepted.
>
> I would have thought sys_indirect would be disliked because
> it looks like an ioctl type multiplexor. Whereas sys_clone_request()
> or /proc/self/clone_unshare simply sets arguments in advance, the
> way /proc/self/attr/current does.

I find the sys_indirect() approach very appealing, in particular
because it is designed and motivated by a non-ioctl multiplexing
and backward compatibility in mind. Like any API it can be abused
and misused, but since it applies to actual system calls and not
obscured ioctl, it is far less likely to become a victim (so to
speak ...).

While I prefer the sys_indirect() (personally I find it elegant,

but it isn't clear that it will be merged), from a technical point
of view any of new system call, sys_indirect() or a 2-step approach
approach - all three seem plausible. The final solution, however
needs to be coordinated with the rest of the kernel developers.

Oren.

>
> -serge
> _____
> 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

---

Subject: Re: Namespaces exhausted CLONE_XXX bits problem
Posted by Dave Hansen on Mon, 14 Jan 2008 21:54:34 GMT
View Forum Message <> Reply to Message

On Mon, 2008-01-14 at 16:36 -0500, Oren Laadan wrote:
> I second the concern of running out of 64 bits of flags. In fact, the
> problem with the flags is likely to be valid outside our context, and
> general to the linux kernel soon. Should we not discuss it there
> too ?

It would be pretty easy to make a new one expandable:

```
 sys_newclone(int len, unsigned long *flags_array)
```

Then you could give it a virtually unlimited number of "unsigned long"s
pointed to by "flags_array".

Plus, the old clone just becomes:

```
sys_oldclone(unsigned long flags)
{
 do_newclone(1, &flags);
}
```

We could validate the flags array address in sys_newclone(), then call
do_newclone().

-- Dave

Subject: Re: Namespaces exhausted CLONE_XXX bits problem
Posted by Cedric Le Goater on Tue, 15 Jan 2008 07:53:00 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Cedric Le Goater (clg@fr.ibm.com):
>> to be more precise :
>>
>> long sys_clone_something(struct clone_something_args args)
>>
>> and
>>
>> long sys_unshare_something(struct unshare_something_args args)
>>
>> The arg passing will be slower bc of the copy_from_user() but we will
>> still have the sys_clone syscall for the fast path.
>>
>> C.
>
> I'm fine with the direction you're going, but just as one more option,
> we could follow more of the selinux/lsm approach of first requesting
> clone/unshare options, then doing the actual clone/unshare. So
> something like
>
> sys_clone_request(extended_64bit_clone_flags)
> sys_clone(usual args)
>
> or
>
> echo pid,mqueue,user,ipc,uts,net > /proc/self/clone_unshare
> clone()

For my information, why selinux/lsm chose that 2 steps approach ?
What kind of issues are they trying to solve ?

Thanks,

C.

Subject: Re: Namespaces exhausted CLONE_XXX bits problem
Posted by Pavel Emelianov on Tue, 15 Jan 2008 08:25:56 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Mon, 2008-01-14 at 16:36 -0500, Oren Laadan wrote:
>> I second the concern of running out of 64 bits of flags. In fact, the
>> problem with the flags is likely to be valid outside our context, and
>> general to the linux kernel soon. Should we not discuss it there
>> too ?
>
> It would be pretty easy to make a new one expandable:
>
> sys_newclone(int len, unsigned long *flags_array)
>
> Then you could give it a virtually unlimited number of "unsigned long"s
> pointed to by "flags_array".
>
> Plus, the old clone just becomes:
>
> sys_oldclone(unsigned long flags)
> {
>   do_newclone(1, &flags);
> }
>
> We could validate the flags array address in sys_newclone(), then call
> do_newclone().

Hmm. I have an idea how to make this w/o a new system call. This might
look wierd, but. Why not stopple the last bit with a CLONE_NEWCLONE and
consider the parent_tidptr/child_tidptr in this case as the pointer to
an array of extra arguments/flargs?

> -- Dave
>
>

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

Subject: Re: Namespaces exhausted CLONE_XXX bits problem
Posted by Cedric Le Goater on Tue, 15 Jan 2008 08:39:50 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:

> Dave Hansen wrote:
>> On Mon, 2008-01-14 at 16:36 -0500, Oren Laadan wrote:
>>> I second the concern of running out of 64 bits of flags. In fact, the
>>> problem with the flags is likely to be valid outside our context, and
>>> general to the linux kernel soon. Should we not discuss it there
>>> too ?
>> It would be pretty easy to make a new one expandable:
>>
>> sys_newclone(int len, unsigned long *flags_array)
>>
>> Then you could give it a virtually unlimited number of "unsigned long"s
>> pointed to by "flags_array".
>>
>> Plus, the old clone just becomes:
>>
>>   sys_oldclone(unsigned long flags)
>>   {
>>   do_newclone(1, &flags);
>>   }
>>
>> We could validate the flags array address in sys_newclone(), then call
>> do_newclone().
>
> Hmm. I have an idea how to make this w/o a new system call. This might
> look wierd, but. Why not stopple the last bit with a CLONE_NEWCLONE and
> consider the parent_tidptr/child_tidptr in this case as the pointer to
> an array of extra arguments/flargs?

It's a bit hacky but it looks like a good idea to me !

Shall we use parent_tidptr or child_tidptr to pass a extended array of
flags only ? if we could pass the pid of the task to be cloned, it would
be useful for c/r.

C.

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

---

Subject: Re: Namespaces exhausted CLONE_XXX bits problem
Posted by Pavel Emelianov on Tue, 15 Jan 2008 08:53:39 GMT
View Forum Message <> Reply to Message

Cedric Le Goater wrote:
> Pavel Emelyanov wrote:
>> Dave Hansen wrote:

>>> On Mon, 2008-01-14 at 16:36 -0500, Oren Laadan wrote:
>>>> I second the concern of running out of 64 bits of flags. In fact, the
>>>> problem with the flags is likely to be valid outside our context, and
>>>> general to the linux kernel soon. Should we not discuss it there
>>>> too ?
>>> It would be pretty easy to make a new one expandable:
>>>
>>> sys_newclone(int len, unsigned long *flags_array)
>>>
>>> Then you could give it a virtually unlimited number of "unsigned long"s
>>> pointed to by "flags_array".
>>>
>>> Plus, the old clone just becomes:
>>>
>>>   sys_oldclone(unsigned long flags)
>>>   {
>>>   do_newclone(1, &flags);
>>>   }
>>>
>>> We could validate the flags array address in sys_newclone(), then call
>>> do_newclone().
>> Hmm. I have an idea how to make this w/o a new system call. This might
>> look wierd, but. Why not stopple the last bit with a CLONE_NEWCLONE and
>> consider the parent_tidptr/child_tidptr in this case as the pointer to
>> an array of extra arguments/flargs?
>
> It's a bit hacky but it looks like a good idea to me !
>
> Shall we use parent_tidptr or child_tidptr to pass a extended array of
> flags only ? if we could pass the pid of the task to be cloned, it would
> be useful for c/r.

Yup. I think we can declare a

```
struct new_clone_arg {
 unsigned int size;
};
```

and consider the xx_tidptr to be a pointer on it. After this we
may sen patches that add fields to this structure.

E.g. first

```
 struct new_clone_arg {
  unsigned int size;
+ unsigned long new_flags;
 };
```

to add flags for cloning new namespaces. Later

```
 struct new_clone_arg {
  unsigned int size;
  unsigned long new_flags;
+ int desired_pid;
 };
```

and each code that needs to access the extra argument would need
to check for new_clone_arg->size to be not less than the offset
of the field he need an access to. E.g. like this:

```
#define clone_arg_has(arg, member) ({   \
 struct new_clone_arg *__carg = arg;   \
 (__carg->size >= offsetof(struct new_clone_arg, member) + \
  sizeof(__carg->member)) })
```

...

```
if (!clone_arg_has(arg, desired_pid))
 return -EINVAL;
```

This would keep the API always compatible.

> C.
>

Thanks,
Pavel

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

---

Subject: Re: Namespaces exhausted CLONE_XXX bits problem
Posted by Dave Hansen on Tue, 15 Jan 2008 09:22:18 GMT
View Forum Message <> Reply to Message

On Tue, 2008-01-15 at 11:25 +0300, Pavel Emelyanov wrote:
> Hmm. I have an idea how to make this w/o a new system call. This might
> look wierd, but. Why not stopple the last bit with a CLONE_NEWCLONE and
> consider the parent_tidptr/child_tidptr in this case as the pointer to
> an array of extra arguments/flargs?

I guess that does keep us from having to add an _actual_ system call.
Do we make the array something like

array[] = { orig_tidptr, nr_flags, actual flags... };

?

-- Dave

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

---

Subject: Re: Namespaces exhausted CLONE_XXX bits problem
Posted by Pavel Emelianov on Tue, 15 Jan 2008 09:24:33 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Tue, 2008-01-15 at 11:25 +0300, Pavel Emelyanov wrote:
>> Hmm. I have an idea how to make this w/o a new system call. This might
>> look wierd, but. Why not stopple the last bit with a CLONE_NEWCLONE and
>> consider the parent_tidptr/child_tidptr in this case as the pointer to
>> an array of extra arguments/flargs?
>
> I guess that does keep us from having to add an _actual_ system call.

Exactly!

> Do we make the array something like
>
> array[] = { orig_tidptr, nr_flags, actual flags... };

Not exactly. I have already sent my view of this in
another letter.

> ?
>
> -- Dave
>
>

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

---

Subject: Re: Namespaces exhausted CLONE_XXX bits problem

Pavel Emelyanov wrote:
> Cedric Le Goater wrote:
>> Pavel Emelyanov wrote:
>>> Dave Hansen wrote:
>>>> On Mon, 2008-01-14 at 16:36 -0500, Oren Laadan wrote:
>>>>> I second the concern of running out of 64 bits of flags. In fact, the
>>>>> problem with the flags is likely to be valid outside our context, and
>>>>> general to the linux kernel soon. Should we not discuss it there
>>>>> too ?
>>>> It would be pretty easy to make a new one expandable:
>>>>
>>>> sys_newclone(int len, unsigned long *flags_array)
>>>>
>>>> Then you could give it a virtually unlimited number of "unsigned long"s
>>>> pointed to by "flags_array".
>>>>
>>>> Plus, the old clone just becomes:
>>>>
>>>> sys_oldclone(unsigned long flags)
>>>> {
>>>>   do_newclone(1, &flags);
>>>> }
>>>>
>>>> We could validate the flags array address in sys_newclone(), then call
>>>> do_newclone().
>>> Hmm. I have an idea how to make this w/o a new system call. This might
>>> look wierd, but. Why not stopple the last bit with a CLONE_NEWCLONE and
>>> consider the parent_tidptr/child_tidptr in this case as the pointer to
>>> an array of extra arguments/flargs?
>> It's a bit hacky but it looks like a good idea to me !
>>
>> Shall we use parent_tidptr or child_tidptr to pass a extended array of
>> flags only ? if we could pass the pid of the task to be cloned, it would
>> be useful for c/r.
>
> Yup. I think we can declare a
>
> struct new_clone_arg {
> unsigned int size;
> };
>
> and consider the xx_tidptr to be a pointer on it. After this we
> may sen patches that add fields to this structure.
>
> E.g. first
>

> struct new_clone_arg {
>   unsigned int size;
> + unsigned long new_flags;
> };
>
> to add flags for cloning new namespaces. Later
>
> struct new_clone_arg {
>   unsigned int size;
>   unsigned long new_flags;
> + int desired_pid;
> };
>
> and each code that needs to access the extra argument would need
> to check for new_clone_arg->size to be not less than the offset
> of the field he need an access to. E.g. like this:
>
> #define clone_arg_has(arg, member) ({   \
> struct new_clone_arg *__carg = arg;   \
> (__carg->size >= offsetof(struct new_clone_arg, member) + \
>   sizeof(__carg->member)) })
>
> ...
>
> if (!clone_arg_has(arg, desired_pid))
> return -EINVAL;
>
> This would keep the API always compatible.

Pavel, this is pretty neat.

I think we need to work on a patch now and send it to andrew and lkml@
to have a larger audience.

I doesn't seem to be a really big patch and I wondering how I could help.
We still have to prepare something for security_task_create()

Thanks !

C.

Subject: Re: Namespaces exhausted CLONE_XXX bits problem

Cedric Le Goater wrote:
> Pavel Emelyanov wrote:
>> Cedric Le Goater wrote:
>>> Pavel Emelyanov wrote:
>>>> Dave Hansen wrote:
>>>>> On Mon, 2008-01-14 at 16:36 -0500, Oren Laadan wrote:
>>>>>> I second the concern of running out of 64 bits of flags. In fact, the
>>>>>> problem with the flags is likely to be valid outside our context, and
>>>>>> general to the linux kernel soon. Should we not discuss it there
>>>>>> too ?
>>>>> It would be pretty easy to make a new one expandable:
>>>>>
>>>>> sys_newclone(int len, unsigned long *flags_array)
>>>>>
>>>>> Then you could give it a virtually unlimited number of "unsigned long"s
>>>>> pointed to by "flags_array".
>>>>>
>>>>> Plus, the old clone just becomes:
>>>>>
>>>>>   sys_oldclone(unsigned long flags)
>>>>>   {
>>>>>   do_newclone(1, &flags);
>>>>>   }
>>>>>
>>>>> We could validate the flags array address in sys_newclone(), then call
>>>>> do_newclone().
>>>> Hmm. I have an idea how to make this w/o a new system call. This might
>>>> look wierd, but. Why not stopple the last bit with a CLONE_NEWCLONE and
>>>> consider the parent_tidptr/child_tidptr in this case as the pointer to
>>>> an array of extra arguments/flargs?
>>> It's a bit hacky but it looks like a good idea to me !
>>>
>>> Shall we use parent_tidptr or child_tidptr to pass a extended array of
>>> flags only ? if we could pass the pid of the task to be cloned, it would
>>> be useful for c/r.
>> Yup. I think we can declare a
>>
>> struct new_clone_arg {
>> unsigned int size;
>> };
>>
>> and consider the xx_tidptr to be a pointer on it. After this we
>> may sen patches that add fields to this structure.
>>
>> E.g. first
>>

>> struct new_clone_arg {
>>   unsigned int size;
>> + unsigned long new_flags;
>> };
>>
>> to add flags for cloning new namespaces. Later
>>
>> struct new_clone_arg {
>>   unsigned int size;
>>   unsigned long new_flags;
>> + int desired_pid;
>> };
>>
>> and each code that needs to access the extra argument would need
>> to check for new_clone_arg->size to be not less than the offset
>> of the field he need an access to. E.g. like this:
>>
>> #define clone_arg_has(arg, member) ({   \
>> struct new_clone_arg *__carg = arg;   \
>> (__carg->size >= offsetof(struct new_clone_arg, member) + \
>>   sizeof(__carg->member)) })
>>
>> ...
>>
>> if (!clone_arg_has(arg, desired_pid))
>> return -EINVAL;
>>
>> This would keep the API always compatible.
>
> Pavel, this is pretty neat.

Thanks, but what to do with unshare()? Stop unsharing namespaces
is not an option, so we'll have to add a new sys_unshare2 system
call with similar technique for argument passing.

> I think we need to work on a patch now and send it to andrew and lkml@
> to have a larger audience.

OK, I'll try to prepare the one for clone() today. Hope it will
be ready to be sent tomorrow.

> I doesn't seem to be a really big patch and I wondering how I could help.

I'll send it for pre-review before showing to people ;)

> We still have to prepare something for security_task_create()
>
> Thanks !

>
> C.

Thanks,
Pavel

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

---

Subject: Re: Namespaces exhausted CLONE_XXX bits problem
Posted by serue on Tue, 15 Jan 2008 14:35:24 GMT
View Forum Message <> Reply to Message

Quoting Cedric Le Goater (clg@fr.ibm.com):
> Serge E. Hallyn wrote:
> > Quoting Cedric Le Goater (clg@fr.ibm.com):
> >> to be more precise :
> >>
> >> long sys_clone_something(struct clone_something_args args)
> >>
> >> and
> >>
> >> long sys_unshare_something(struct unshare_something_args args)
> >>
> >> The arg passing will be slower bc of the copy_from_user() but we will
> >> still have the sys_clone syscall for the fast path.
> >>
> >> C.
> >
> > I'm fine with the direction you're going, but just as one more option,
> > we could follow more of the selinux/lsm approach of first requesting
> > clone/unshare options, then doing the actual clone/unshare. So
> > something like
> >
> > sys_clone_request(extended_64bit_clone_flags)
> > sys_clone(usual args)
> >
> > or
> >
> > echo pid,mqueue,user,ipc,uts,net > /proc/self/clone_unshare
> > clone()
>
> For my information, why selinux/lsm chose that 2 steps approach ?
> What kind of issues are they trying to solve ?

Well an interface was needed to allow multiple LSMs to query and set

task information. Using a syscall (which was attempted) required
ioctl-like subcommands which was not accepted.

-serge

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

---

Subject: Re: Namespaces exhausted CLONE_XXX bits problem
Posted by serue on Tue, 15 Jan 2008 15:08:06 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> Dave Hansen wrote:
> > On Tue, 2008-01-15 at 11:25 +0300, Pavel Emelyanov wrote:
> >> Hmm. I have an idea how to make this w/o a new system call. This might
> >> look wierd, but. Why not stopple the last bit with a CLONE_NEWCLONE and
> >> consider the parent_tidptr/child_tidptr in this case as the pointer to
> >> an array of extra arguments/flargs?
> >
> > I guess that does keep us from having to add an _actual_ system call.
>
> Exactly!

I'll be honest, while it's a really neat idea, in terms of code actually
going into tree I far far prefer a real new syscall.

But it sounds like I'm the only one so I'll just mention it once and
then bite my tongue :)

thanks,
-serge

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

---

Subject: Re: Namespaces exhausted CLONE_XXX bits problem
Posted by Cedric Le Goater on Tue, 15 Jan 2008 15:51:50 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelyanov (xemul@openvz.org):
>> Dave Hansen wrote:

>>> On Tue, 2008-01-15 at 11:25 +0300, Pavel Emelyanov wrote:
>>>> Hmm. I have an idea how to make this w/o a new system call. This might
>>>> look wierd, but. Why not stopple the last bit with a CLONE_NEWCLONE and
>>>> consider the parent_tidptr/child_tidptr in this case as the pointer to
>>>> an array of extra arguments/flargs?
>>> I guess that does keep us from having to add an _actual_ system call.
>> Exactly!
>
> I'll be honest, while it's a really neat idea, in terms of code actually
> going into tree I far far prefer a real new syscall.

well, hijacking child_tidptr and adding a new syscall will probably look
the same internally. so if it ends up that hijacking child_tidptr is not
acceptable, we won't have much work to plug it in a new syscall.

> But it sounds like I'm the only one so I'll just mention it once and
> then bite my tongue :)

hold on. this patch has not been sent on lkml@ but it's worth a try :)

C.

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