
Subject: [RFC][PATCH] Make access to taks's nsproxy liter
Posted by [Pavel Emelianov](#) on Wed, 08 Aug 2007 15:37:05 GMT
[View Forum Message](#) <> [Reply to Message](#)

When someone wants to deal with some other taks's namespaces it has to lock the task and then to get the desired namespace if the one exists. This is slow on read-only paths and may be impossible in some cases.

E.g. Oleg recently noticed a race between unshare() and the (just sent for review) pid namespaces - when the task notifies the parent it has to know the parent's namespace, but taking the task_lock() is impossible there - the code is under write locked tasklist lock.

On the other hand switching the namespace on task (daemonize) and releasing the namespace (after the last task exit) is rather rare operation and we can sacrifice its speed to solve the issues above.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
fs/proc/base.c      | 27 ++++++-----
include/linux/nsproxy.h | 19 +++++-----
kernel/exit.c        | 4 +---
kernel/fork.c         | 11 +++++-----
kernel/nsproxy.c      | 38 ++++++-----
5 files changed, 60 insertions(+), 39 deletions(-)
```

```
diff --git a/fs/proc/base.c b/fs/proc/base.c
index ed2b224..614851a 100644
--- a/fs/proc/base.c
+++ b/fs/proc/base.c
@@ -371,18 +371,21 @@ struct proc_mounts {
 static int mounts_open(struct inode *inode, struct file *file)
 {
     struct task_struct *task = get_proc_task(inode);
+ struct nsproxy *nsp;
     struct mnt_namespace *ns = NULL;
     struct proc_mounts *p;
     int ret = -EINVAL;

     if (task) {
- task_lock(task);
- if (task->nsproxy) {
- ns = task->nsproxy->mnt_ns;
```

```

+ rcu_read_lock();
+ nsp = task_nsproxy(task);
+ if (nsp) {
+   ns = nsp->mnt_ns;
+   if (ns)
+     get_mnt_ns(ns);
+ }
- task_unlock(task);
+ rcu_read_unlock();
+
+   put_task_struct(task);
+ }

```

@@ -445,16 +448,20 @@ static int mountstats_open(struct inode

```

if (!ret) {
    struct seq_file *m = file->private_data;
+   struct nsproxy *nsp;
    struct mnt_namespace *mnt_ns = NULL;
    struct task_struct *task = get_proc_task(inode);

    if (task) {
-   task_lock(task);
-   if (task->nsproxy)
-     mnt_ns = task->nsproxy->mnt_ns;
-   if (mnt_ns)
-     get_mnt_ns(mnt_ns);
-   task_unlock(task);
+   rcu_read_lock();
+   nsp = task_nsproxy(task);
+   if (nsp) {
+     mnt_ns = nsp->mnt_ns;
+     if (mnt_ns)
+       get_mnt_ns(mnt_ns);
+   }
+   rcu_read_unlock();
+
+   put_task_struct(task);
+ }

```

diff --git a/include/linux/nsproxy.h b/include/linux/nsproxy.h

index 525d8fc..74f21fe 100644

--- a/include/linux/nsproxy.h

+++ b/include/linux/nsproxy.h

```

@@ -32,8 +32,14 @@ struct nsproxy {
};
extern struct nsproxy init_nsproxy;

```

```

+static inline struct nsproxy *task_nsproxy(struct task_struct *tsk)
+{
+ return rcu_dereference(tsk->nsproxy);
+}
+
int copy_namespaces(unsigned long flags, struct task_struct *tsk);
-void get_task_namespaces(struct task_struct *tsk);
+void exit_task_namespaces(struct task_struct *tsk);
+void switch_task_namespaces(struct task_struct *tsk, struct nsproxy *new);
void free_nsproxy(struct nsproxy *ns);
int unshare_nsproxy_namespaces(unsigned long, struct nsproxy **,
    struct fs_struct *);
@@ -45,17 +51,6 @@ static inline void put_nsproxy(struct ns
    }
}

-static inline void exit_task_namespaces(struct task_struct *p)
-{
- struct nsproxy *ns = p->nsproxy;
- if (ns) {
- task_lock(p);
- p->nsproxy = NULL;
- task_unlock(p);
- put_nsproxy(ns);
- }
-}
-
#ifdef CONFIG_CONTAINER_NS
int ns_container_clone(struct task_struct *tsk);
#else
diff --git a/kernel/exit.c b/kernel/exit.c
index 2f4f35e..a2aef1f 100644
--- a/kernel/exit.c
+++ b/kernel/exit.c
@@ -409,9 +409,7 @@ void daemonize(const char *name, ...)
    current->fs = fs;
    atomic_inc(&fs->count);

- exit_task_namespaces(current);
- current->nsproxy = init_task.nsproxy;
- get_task_namespaces(current);
+ switch_task_namespaces(current, init_task.nsproxy);

    exit_files(current);
    current->files = init_task.files;
diff --git a/kernel/fork.c b/kernel/fork.c
index bc48e0f..0c6dd67 100644
--- a/kernel/fork.c

```

```

+++ b/kernel/fork.c
@@ -1618,7 +1618,7 @@ asmlinkage long sys_unshare(unsigned lon
    struct mm_struct *mm, *new_mm = NULL, *active_mm = NULL;
    struct files_struct *fd, *new_fd = NULL;
    struct sem_undo_list *new_ulist = NULL;
- struct nsproxy *new_nsproxy = NULL, *old_nsproxy = NULL;
+ struct nsproxy *new_nsproxy = NULL;

    check_unshare_flags(&unshare_flags);

@@ -1647,14 +1647,13 @@ asmlinkage long sys_unshare(unsigned lon

    if (new_fs || new_mm || new_fd || new_ulist || new_nsproxy) {

- task_lock(current);
-
    if (new_nsproxy) {
- old_nsproxy = current->nsproxy;
- current->nsproxy = new_nsproxy;
- new_nsproxy = old_nsproxy;
+ switch_task_namespaces(current, new_nsproxy);
+ new_nsproxy = NULL;
    }

+ task_lock(current);
+
    if (new_fs) {
        fs = current->fs;
        current->fs = new_fs;
diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index 58843ae..91dca27 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -30,14 +30,6 @@ static inline void get_nsproxy(struct ns
    atomic_inc(&ns->count);
}

-void get_task_namespaces(struct task_struct *tsk)
-{
- struct nsproxy *ns = tsk->nsproxy;
- if (ns) {
- get_nsproxy(ns);
- }
-}
-
/*
 * creates a copy of "orig" with refcount 1.
 */

```

@@ -205,6 +197,36 @@ out:

```
    return err;
}

+void switch_task_namespaces(struct task_struct *p, struct nsproxy *new)
+{
+ struct nsproxy *ns;
+
+ might_sleep();
+
+ ns = p->nsproxy;
+ if (ns == new)
+ return;
+
+ if (new)
+ get_nsproxy(new);
+ rcu_assign_pointer(p->nsproxy, new);
+
+ if (ns && atomic_dec_and_test(&ns->count)) {
+ /*
+  * wait for others to get what they want from this
+  * nsproxy. cannot release this nsproxy via the
+  * call_rcu() since put_mnt_ns will want to sleep
+  */
+ synchronize_rcu();
+ free_nsproxy(ns);
+ }
+}
+
+void exit_task_namespaces(struct task_struct *p)
+{
+ switch_task_namespaces(p, NULL);
+}
+
+static int __init nsproxy_cache_init(void)
+{
+ nsproxy_cachep = kmem_cache_create("nsproxy", sizeof(struct nsproxy),
```

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>

Subject: Re: [RFC][PATCH] Make access to taks's nsproxy liter

Posted by [Oleg Nesterov](#) on Wed, 08 Aug 2007 16:37:57 GMT

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

On 08/08, Pavel Emelyanov wrote:

>
 > When someone wants to deal with some other tasks's namespaces
 > it has to lock the task and then to get the desired namespace
 > if the one exists. This is slow on read-only paths and may be
 > impossible in some cases.
 >
 > E.g. Oleg recently noticed a race between unshare() and the
 > (just sent for review) pid namespaces - when the task notifies
 > the parent it has to know the parent's namespace, but taking
 > the task_lock() is impossible there - the code is under write
 > locked tasklist lock.
 >
 > On the other hand switching the namespace on task (daemonize)
 > and releasing the namespace (after the last task exit) is rather
 > rare operation and we can sacrifice its speed to solve the
 > issues above.

Still it is a bit sad we slow down process's exit. Perhaps I missed
 some other ->nsproxy access, but can't we make a simpler patch?

```
--- kernel/fork.c 2007-07-28 16:58:17.000000000 +0400
+++ /proc/self/fd/0 2007-08-08 20:30:33.325216944 +0400
@@ -1633,7 +1633,9 @@ asmlinkage long sys_unshare(unsigned lon
```

```
    if (new_nsproxy) {
        old_nsproxy = current->nsproxy;
+   read_lock(&tasklist_lock);
        current->nsproxy = new_nsproxy;
+   read_unlock(&tasklist_lock);
        new_nsproxy = old_nsproxy;
    }
```

This way ->nsproxy is stable under task_lock() or write_lock(tasklist).

```
> +void switch_task_namespaces(struct task_struct *p, struct nsproxy *new)
> +{
> + struct nsproxy *ns;
> +
> + might_sleep();
> +
> + ns = p->nsproxy;
> + if (ns == new)
> + return;
> +
> + if (new)
> + get_nsproxy(new);
> + rcu_assign_pointer(p->nsproxy, new);
```

```
> +
> + if (ns && atomic_dec_and_test(&ns->count)) {
> + /*
> +  * wait for others to get what they want from this
> +  * nsproxy. cannot release this nsproxy via the
> +  * call_rcu() since put_mnt_ns will want to sleep
> +  */
> + synchronize_rcu();
> + free_nsproxy(ns);
> + }
> +}
```

(I may be wrong, Paul cc'ed)

This is correct with the current implementation of RCU, but strictly speaking, we can't use `synchronize_rcu()` here, because `write_lock_irq()` doesn't imply `rcu_read_lock()` in theory.

Oleg.

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

Subject: Re: [RFC][PATCH] Make access to taks's nsproxy liter
Posted by [Oleg Nesterov](#) on Wed, 08 Aug 2007 16:41:07 GMT
[View Forum Message](#) <> [Reply to Message](#)

This time Paul E. McKenney actually cc'ed, sorry for the extra noise...

On 08/08, Pavel Emelyanov wrote:

```
>
> When someone wants to deal with some other taks's namespaces
> it has to lock the task and then to get the desired namespace
> if the one exists. This is slow on read-only paths and may be
> impossible in some cases.
>
> E.g. Oleg recently noticed a race between unshare() and the
> (just sent for review) pid namespaces - when the task notifies
> the parent it has to know the parent's namespace, but taking
> the task_lock() is impossible there - the code is under write
> locked tasklist lock.
>
> On the other hand switching the namespace on task (daemonize)
> and releasing the namespace (after the last task exit) is rather
```

> rare operation and we can sacrifice its speed to solve the
> issues above.

Still it is a bit sad we slow down process's exit. Perhaps I missed
some other ->nsproxy access, but can't we make a simpler patch?

```
--- kernel/fork.c 2007-07-28 16:58:17.000000000 +0400
+++ /proc/self/fd/0 2007-08-08 20:30:33.325216944 +0400
@@ -1633,7 +1633,9 @@ asmlinkage long sys_unshare(unsigned lon
```

```
    if (new_nsproxy) {
        old_nsproxy = current->nsproxy;
+   read_lock(&tasklist_lock);
        current->nsproxy = new_nsproxy;
+   read_unlock(&tasklist_lock);
        new_nsproxy = old_nsproxy;
    }
```

This way ->nsproxy is stable under task_lock() or write_lock(tasklist).

```
> +void switch_task_namespaces(struct task_struct *p, struct nsproxy *new)
> +{
> + struct nsproxy *ns;
> +
> + might_sleep();
> +
> + ns = p->nsproxy;
> + if (ns == new)
> + return;
> +
> + if (new)
> + get_nsproxy(new);
> + rcu_assign_pointer(p->nsproxy, new);
> +
> + if (ns && atomic_dec_and_test(&ns->count)) {
> + /*
> +  * wait for others to get what they want from this
> +  * nsproxy. cannot release this nsproxy via the
> +  * call_rcu() since put_mnt_ns will want to sleep
> +  */
> + synchronize_rcu();
> + free_nsproxy(ns);
> + }
> +}
```

(I may be wrong, Paul cc'ed)

This is correct with the current implementation of RCU, but strictly speaking, we can't use `synchronize_rcu()` here, because `write_lock_irq()` doesn't imply `rcu_read_lock()` in theory.

Oleg.

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>

Subject: Re: [RFC][PATCH] Make access to taks's nsproxy liter
Posted by [Pavel Emelianov](#) on Thu, 09 Aug 2007 07:14:18 GMT
[View Forum Message](#) <> [Reply to Message](#)

Oleg Nesterov wrote:

> This time Paul E. McKenney actually cc'ed, sorry for the extra
> noise...

>

> On 08/08, Pavel Emelyanov wrote:

>> When someone wants to deal with some other taks's namespaces
>> it has to lock the task and then to get the desired namespace
>> if the one exists. This is slow on read-only paths and may be
>> impossible in some cases.

>>

>> E.g. Oleg recently noticed a race between `unshare()` and the
>> (just sent for review) pid namespaces - when the task notifies
>> the parent it has to know the parent's namespace, but taking
>> the `task_lock()` is impossible there - the code is under write
>> locked tasklist lock.

>>

>> On the other hand switching the namespace on task (`daemonize`)
>> and releasing the namespace (after the last task exit) is rather
>> rare operation and we can sacrifice its speed to solve the
>> issues above.

>

> Still it is a bit sad we slow down process's exit. Perhaps I missed
> some other `->nsproxy` access, but can't we make a simpler patch?

>

> --- kernel/fork.c 2007-07-28 16:58:17.000000000 +0400

> +++ /proc/self/fd/0 2007-08-08 20:30:33.325216944 +0400

> @@ -1633,7 +1633,9 @@ asmlinkage long sys_unshare(unsigned lon

>

> if (new_nsproxy) {
> old_nsproxy = current->nsproxy;
> + read_lock(&tasklist_lock);
> current->nsproxy = new_nsproxy;

```
> + read_unlock(&tasklist_lock);
>   new_nsproxy = old_nsproxy;
> }
>
>
> This way ->nsproxy is stable under task_lock() or write_lock(tasklist).
```

We may, but the intention of this patch is just (!) to make the access to other's task namespaces lockless. Solving the accessing parent's nsproxy in do_notify_parent() is a (good) side effect :)

```
>> +void switch_task_namespaces(struct task_struct *p, struct nsproxy *new)
>> +{
>> + struct nsproxy *ns;
>> +
>> + might_sleep();
>> +
>> + ns = p->nsproxy;
>> + if (ns == new)
>> + return;
>> +
>> + if (new)
>> + get_nsproxy(new);
>> + rcu_assign_pointer(p->nsproxy, new);
>> +
>> + if (ns && atomic_dec_and_test(&ns->count)) {
>> + /*
>> +  * wait for others to get what they want from this
>> +  * nsproxy. cannot release this nsproxy via the
>> +  * call_rcu() since put_mnt_ns will want to sleep
>> +  */
>> + synchronize_rcu();
>> + free_nsproxy(ns);
>> + }
>> +}
>
> (I may be wrong, Paul cc'ed)
>
> This is correct with the current implementation of RCU, but strictly speaking,
> we can't use synchronize_rcu() here, because write_lock_irq() doesn't imply
> rcu_read_lock() in theory.
>
> Oleg.
>
>
```

Containers mailing list

Subject: Re: [RFC][PATCH] Make access to taks's nsproxy liter
Posted by [ebiederm](#) on Thu, 09 Aug 2007 08:00:19 GMT
[View Forum Message](#) <> [Reply to Message](#)

Pavel Emelyanov <xemul@openvz.org> writes:

> Eric W. Biederman wrote:
>> Pavel Emelyanov <xemul@openvz.org> writes:
>>
>>> When someone wants to deal with some other taks's namespaces
>>> it has to lock the task and then to get the desired namespace
>>> if the one exists. This is slow on read-only paths and may be
>>> impossible in some cases.
>>>
>>> E.g. Oleg recently noticed a race between unshare() and the
>>> (just sent for review) pid namespaces - when the task notifies
>>> the parent it has to know the parent's namespace, but taking
>>> the task_lock() is impossible there - the code is under write
>>> locked tasklist lock.
>>>
>>> On the other hand switching the namespace on task (daemonize)
>>> and releasing the namespace (after the last task exit) is rather
>>> rare operation and we can sacrifice its speed to solve the
>>> issues above.
>>
>>
>> Looks pretty decent but we need to clearly say what lock
>> you have to have to assign to the nsproxy pointer.
>
> no locks - only current is allowed to change its nsproxy!

Exactly. Just need a comment or something to make that clear.

Eric

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

Subject: Re: [RFC][PATCH] Make access to taks's nsproxy liter
Posted by [Pavel Emelianov](#) on Thu, 09 Aug 2007 08:17:08 GMT
[View Forum Message](#) <> [Reply to Message](#)

Eric W. Biederman wrote:

> Pavel Emelyanov <xemul@openvz.org> writes:

>

>> Eric W. Biederman wrote:

>>> Pavel Emelyanov <xemul@openvz.org> writes:

>>>

>>>> When someone wants to deal with some other tasks's namespaces

>>>> it has to lock the task and then to get the desired namespace

>>>> if the one exists. This is slow on read-only paths and may be

>>>> impossible in some cases.

>>>>

>>>> E.g. Oleg recently noticed a race between unshare() and the

>>>> (just sent for review) pid namespaces - when the task notifies

>>>> the parent it has to know the parent's namespace, but taking

>>>> the task_lock() is impossible there - the code is under write

>>>> locked tasklist lock.

>>>>

>>>> On the other hand switching the namespace on task (daemonize)

>>>> and releasing the namespace (after the last task exit) is rather

>>>> rare operation and we can sacrifice its speed to solve the

>>>> issues above.

>>>

>>> Looks pretty decent but we need to clearly say what lock

>>> you have to have to assign to the nsproxy pointer.

>> no locks - only current is allowed to change its nsproxy!

>

> Exactly. Just need a comment or something to make that clear.

Oh! Ok, I will add it. Thanks.

> Eric

>

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>

Subject: Re: [RFC][PATCH] Make access to tasks's nsproxy liter

Posted by [serue](#) on Thu, 09 Aug 2007 14:10:19 GMT

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

Quoting Pavel Emelyanov (xemul@openvz.org):

> [snip]

>

> >>diff --git a/include/linux/nsproxy.h b/include/linux/nsproxy.h

> >>index 525d8fc..74f21fe 100644

```

>>--- a/include/linux/nsproxy.h
>>+++ b/include/linux/nsproxy.h
>>@@ -32,8 +32,14 @@ struct nsproxy {
>>};
>>extern struct nsproxy init_nsproxy;
>>
>>+static inline struct nsproxy *task_nsproxy(struct task_struct *tsk)
>>+{
>>+ return rcu_dereference(tsk->nsproxy);
>>+}
>>
>>Looks like a very nice cleanup as well. But please add a comment
>>above task_nsproxy() that it must be called under rcu_read_lock()
>>or task_lock(task) (though I'll admit the rcu_dereference may make that
>>obvious)
>
> I will, but I think that rcu_dereference implies this. Anyway.

```

Yeah... as i said... but I still get people asking "what is rcu anyway" so don't think we can assume the implication is clear to everyone.

```

> [snip]
>
>>+ if (ns == new)
>>+ return;
>>+
>>+ if (new)
>>+ get_nsproxy(new);
>>+ rcu_assign_pointer(p->nsproxy, new);
>>+
>>+ if (ns && atomic_dec_and_test(&ns->count)) {
>>+ /*
>>+  * wait for others to get what they want from this
>>+  * nsproxy. cannot release this nsproxy via the
>>+  * call_rcu() since put_mnt_ns will want to sleep
>>+  */
>>+ synchronize_rcu();
>>+ free_nsproxy(ns);
>>+ }
>>+}
>>
>>Also a comment above switch_task_namespaces() that it must be called
>>with task_lock held.
>
> no! no locks here! free_nsproxy() may sleep when putting mnt_ns and
> maybe some other. see - there's a hunk in sys_unshare that move the
> task_lock() after switch_task_namespaces().

```

Hmm,

Yes, I see. And in the current usages it's correct. I just worry that the function takes the `task_struct` as an arg (which I know it must for `do_exit()`) and so someone else might use it unlocked to switch another task's namespace, for instance in an attempt to implement namespace entering.

So ok "a comment .. that it must be called with `task_lock` held" would be wrong, but please do add the **correct** comment :) Namely that apart from current usage in `do_exit`, this should not be used to switch `nsproxy` on any task but current. (Unless I'm still mistaken of course)

thanks,
-serge

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