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 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.