
Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by [Oleg Nesterov](#) on Fri, 27 Jul 2007 08:07:58 GMT
[View Forum Message](#) <> [Reply to Message](#)

On 07/26, Oleg Nesterov wrote:

>

> On 07/26, Pavel Emelyanov wrote:

> >

> > Make task release its namespaces after it has reparented all his
> > children to child_reaper, but before it notifies its parent about
> > its death.

> >

> > The reason to release namespaces after reparenting is that when task
> > exits it may send a signal to its parent (SIGCHLD), but if the parent
> > has already exited its namespaces there will be no way to decide what
> > pid to defer to him - parent can be from different namespace.

> >

> > The reason to release namespace before notifying the parent is that
> > when task sends a SIGCHLD to parent it can call wait() on this task
> > and release it. But releasing the mnt namespace implies dropping
> > of all the mounts in the mnt namespace and NFS expects the task to
> > have valid sighand pointer.

> >

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

> >

> > ---

> >

> > exit.c | 5 ++++-

> > 1 files changed, 4 insertions(+), 1 deletion(-)

> >

> > diff -upr linux-2.6.23-rc1-mm1.orig/kernel/exit.c

> > linux-2.6.23-rc1-mm1-7/kernel/exit.c

> > --- linux-2.6.23-rc1-mm1.orig/kernel/exit.c 2007-07-26

> > 16:34:45.000000000 +0400

> > +++ linux-2.6.23-rc1-mm1-7/kernel/exit.c 2007-07-26

> > 16:36:37.000000000 +0400

> > @@ -788,6 +804,10 @@ static void exit_notify(struct task_stru

> > BUG_ON(!list_empty(&tsk->children));

> > BUG_ON(!list_empty(&tsk->ptrace_children));

> >

> > + write_unlock_irq(&tasklist_lock);

> > + exit_task_namespaces(tsk);

> > + write_lock_irq(&tasklist_lock);

>

> No.

>

> We "cleared" our ->children/->ptrace_children lists. Now suppose that
> another thread dies, and its forget_original_parent() choose us as a

> new reaper before we re-take tasklist.

Perhaps, we can do something like the patch below. Roland, what do you think?

We can check PF_EXITING instead of ->exit_state while choosing the new parent. Note that tasklist_lock acts as a barrier, everyone who takes tasklist after us (when forget_original_parent() drops it) must see PF_EXITING.

Oleg.

```
--- t/kernel/exit.c~ 2007-07-27 11:32:21.000000000 +0400
+++ t/kernel/exit.c 2007-07-27 11:59:09.000000000 +0400
@@ -686,11 +686,14 @@ reparent_thread(struct task_struct *p, s
 * the child reaper process (ie "init") in our pid
 * space.
 */
-static void
-forget_original_parent(struct task_struct *father, struct list_head *to_release)
+static void forget_original_parent(struct task_struct *father)
{
    struct task_struct *p, *reaper = father;
- struct list_head *_p, *_n;
+ struct list_head *ptrace_dead, *_p, *_n;
+
+ INIT_LIST_HEAD(&ptrace_dead);
+
+ write_lock_irq(&tasklist_lock);

    do {
        reaper = next_thread(reaper);
@@ -698,7 +701,7 @@ forget_original_parent(struct task_struct
        reaper = child_reaper(father);
        break;
    }
- } while (reaper->exit_state);
+ } while (reaper->flags & PF_EXITING);

    /*
     * There are only two places where our children can be:
@@ -736,13 +739,25 @@ forget_original_parent(struct task_struct
     * while it was being traced by us, to be able to see it in wait4.
     */
    if (unlikely(ptrace && p->exit_state == EXIT_ZOMBIE && p->exit_signal == -1))
- list_add(&p->ptrace_list, to_release);
+ list_add(&p->ptrace_list, &ptrace_dead);
}
```

```

+
list_for_each_safe(_p, _n, &father->ptrace_children) {
    p = list_entry(_p, struct task_struct, ptrace_list);
    choose_new_parent(p, reaper);
    reparent_thread(p, father, 1);
}
+
+ write_unlock_irq(&tasklist_lock);
+ BUG_ON(!list_empty(&tsk->children));
+ BUG_ON(!list_empty(&tsk->ptrace_children));
+
+ list_for_each_safe(_p, _n, &ptrace_dead) {
+     list_del_init(_p);
+     t = list_entry(_p, struct task_struct, ptrace_list);
+     release_task(t);
+ }
+
}

/*
@@ -753,7 +768,6 @@ static void exit_notify(struct task_stru
{
    int state;
    struct task_struct *t;
- struct list_head ptrace_dead, *_p, *_n;
    struct pid *pgrp;

    if (signal_pending(tsk) && !(tsk->signal->flags & SIGNAL_GROUP_EXIT)
@@ -776,8 +790,6 @@ static void exit_notify(struct task_stru
    read_unlock(&tasklist_lock);
}

- write_lock_irq(&tasklist_lock);
-
/*
 * This does two things:
 *
@@ -786,12 +798,9 @@ static void exit_notify(struct task_stru
 * as a result of our exiting, and if they have any stopped
 * jobs, send them a SIGHUP and then a SIGCONT. (POSIX 3.2.2.2)
 */
+ forget_original_parent(tsk);

- INIT_LIST_HEAD(&ptrace_dead);
- forget_original_parent(tsk, &ptrace_dead);
- BUG_ON(!list_empty(&tsk->children));
- BUG_ON(!list_empty(&tsk->ptrace_children));
-

```

```

+ write_lock_irq(&tasklist_lock);
/*
 * Check to see if any process groups have become orphaned
 * as a result of our exiting, and if they have any stopped
@@ -801,9 +810,8 @@ static void exit_notify(struct task_stru
 * and we were the only connection outside, so our pgrp
 * is about to become orphaned.
 */
-
- t = tsk->real_parent;
-
+
pgrp = task_pgrp(tsk);
if ((task_pgrp(t) != pgrp) &&
    (task_session(t) == task_session(tsk)) &&
@@ -826,9 +834,8 @@ static void exit_notify(struct task_stru
 * If our self_exec id doesn't match our parent_exec_id then
 * we have changed execution domain as these two values started
 * the same after a fork.
- *
- */
-
+
if (tsk->exit_signal != SIGCHLD && tsk->exit_signal != -1 &&
    (tsk->parent_exec_id != t->self_exec_id ||
     tsk->self_exec_id != tsk->parent_exec_id)
@@ -856,12 +863,6 @@ static void exit_notify(struct task_stru

write_unlock_irq(&tasklist_lock);

- list_for_each_safe(_p, _n, &ptrace_dead) {
- list_del_init(_p);
- t = list_entry(_p, struct task_struct, ptrace_list);
- release_task(t);
- }
-
/* If the process is dead, release it - nobody will wait for it */
if (state == EXIT_DEAD)
release_task(tsk);

```
