Subject: [PATCH 1/20] Reowrk forget_original_parent()
Posted by Pavel Emelianov on Tue, 07 Aug 2007 09:29:36 GMT
View Forum Message <> Reply to Message

From: Oleg Nesterov <oleg@tv-sign.ru>

We have to call exit_task_namespaces() only after the exiting task has
reparented all his children and is sure that no other threads will
reparent theirs for it. Why this is needed is explained in appropriate
patch. This one only reworks the forget_original_parent() so that after
this task cannot be/become parent of any other task.

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

The other changes are just cleanups. They just move some code from exit_notify
to forget_original_parent(). It is a bit silly to declare ptrace_dead in
exit_notify(), take tasklist, pass ptrace_dead to forget_original_parent(),
unlock-lock-unlock tasklist, and then use ptrace_dead.

Signed-off-by: Oleg Nesterov <oleg@tv-sign.ru>
Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
---

 exit.c |   39 ++++++++++++++++++++++------------------
 1 files changed, 21 insertions(+), 18 deletions(-)

--- ./kernel/exit.c.ve1 2007-07-27 12:48:59.000000000 +0400
+++ ./kernel/exit.c 2007-07-27 12:51:50.000000000 +0400
@@ -685,10 +685,14 @@ 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, *n, *reaper = father;
+ struct list_head ptrace_dead;
+
+ INIT_LIST_HEAD(&ptrace_dead);
+
+ write_lock_irq(&tasklist_lock);

  do {
   reaper = next_thread(reaper);
```

```
@@ -696,7 +700,7 @@ forget_original_parent(struct task_struc
    reaper = task_child_reaper(father);
    break;
   }
- } while (reaper->exit_state);
+ } while (reaper->flags & PF_EXITING);

  /*
   * There are only two places where our children can be:
@@ -733,12 +737,23 @@ forget_original_parent(struct task_struc
    * 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_entry_safe(p, n, &father->ptrace_children, ptrace_list) {
   choose_new_parent(p, reaper);
   reparent_thread(p, father, 1);
  }
+
+ write_unlock_irq(&tasklist_lock);
+ BUG_ON(!list_empty(&father->children));
+ BUG_ON(!list_empty(&father->ptrace_children));
+
+ list_for_each_entry_safe(p, n, &ptrace_dead, ptrace_list) {
+  list_del_init(&p->ptrace_list);
+  release_task(p);
+ }
+
 }

 /*
@@ -749,7 +764,6 @@ 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)
@@ -772,8 +786,6 @@ static void exit_notify(struct task_stru
   read_unlock(&tasklist_lock);
  }

- write_lock_irq(&tasklist_lock);
-
```

```
  /*
   * This does two things:
   *
@@ -782,12 +794,9 @@ 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
@@ -852,12 +861,6 @@ 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);
```