Subject: [RFC][PATCH] Isolate some explicit usage of task->tgid
Posted by Pavel Emelianov on Fri, 17 Aug 2007 12:39:13 GMT
View Forum Message <> Reply to Message

With pid namespaces this field is now dangerous to use explicitly, so
try to hide it behind helpers. This is not a complete list of users,
but just the core.

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

---

plans:
  the pid, tgid and pgrp fields o task_struct and signal_struct are
  to be marked as deprecated. Unfortunately this patch cannot be
  sent right now as this leads to tons of warnings, so start isolating
  them, and deprecate later to make other maintainers switch to
  using namespace-safe routines.

 fs/exec.c | 4 ++--
 fs/proc/base.c | 2 +-
 include/linux/sched.h | 6 ++++++
 kernel/posix-cpu-timers.c | 12 ++++++++++++------
 kernel/posix-timers.c | 4 ++--
 kernel/ptrace.c | 2 +-
 kernel/signal.c | 2 +-
 mm/oom_kill.c | 2 +-
 8 files changed, 20 insertions(+), 14 deletions(-)

diff --git a/fs/exec.c b/fs/exec.c
index 35013f2..796453b 100644
--- a/fs/exec.c
+++ b/fs/exec.c
@@ -865,8 +865,8 @@ static int de_thread(struct task_struct

   write_lock_irq(&tasklist_lock);

- BUG_ON(leader->tgid != tsk->tgid);
- BUG_ON(tsk->pid == tsk->tgid);
+ BUG_ON(!same_thread_group(leader, tsk));
+ BUG_ON(has_group_leader_pid(tsk));
   /*
    * An exec() starts a new thread group with the
    * TGID of the previous thread group. Rehash the
diff --git a/fs/proc/base.c b/fs/proc/base.c
index e3009ab..6dce57c 100644
--- a/fs/proc/base.c
+++ b/fs/proc/base.c

```
@@ -2474,7 +2474,7 @@ static struct dentry *proc_task_lookup(s
  rcu_read_unlock();
  if (!task)
   goto out;
- if (leader->tgid != task->tgid)
+ if (!same_thread_group(leader, task))
   goto out_drop_task;

  result = proc_task_instantiate(dir, dentry, task, NULL);
diff --git a/include/linux/sched.h b/include/linux/sched.h
index f8cebf8..00a292e 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -1691,6 +1691,12 @@ static inline int has_group_leader_pid(s
  return p->pid == p->tgid;
 }

+static inline
+int same_thread_group(struct task_struct p1, struct task_struct *p2)
+{
+ return p1->tgid == p2->tgid;
+}
+
 static inline struct task_struct *next_thread(const struct task_struct *p)
 {
  return list_entry(rcu_dereference(p->thread_group.next),
diff --git a/kernel/posix-cpu-timers.c b/kernel/posix-cpu-timers.c
index b53c8fc..06a1e7d 100644
--- a/kernel/posix-cpu-timers.c
+++ b/kernel/posix-cpu-timers.c
@@ -21,8 +21,8 @@ static int check_clock(const clockid_t w

  read_lock(&tasklist_lock);
  p = find_task_by_pid(pid);
- if (!p || (CPUCLOCK_PERTHREAD(which_clock) ?
-     p->tgid != current->tgid : p->tgid != pid)) {
+ if (!p || !(CPUCLOCK_PERTHREAD(which_clock) ?
+     same_thread_group(p, current) : has_group_leader_pid(p))) {
   error = -EINVAL;
  }
  read_unlock(&tasklist_lock);
@@ -308,13 +308,13 @@ int posix_cpu_clock_get(const clockid_t
   p = find_task_by_pid(pid);
   if (p) {
    if (CPUCLOCK_PERTHREAD(which_clock)) {
-    if (p->tgid == current->tgid) {
+    if (same_thread_group(p, current)) {
      error = cpu_clock_sample(which_clock,
```

```
          p, &rtn);
     }
    } else {
     read_lock(&tasklist_lock);
-    if (p->tgid == pid && p->signal) {
+    if (has_group_leader_pid(p) && p->signal) {
      error =
          cpu_clock_sample_group(which_clock,
                 p, &rtn);
@@ -355,7 +355,7 @@ int posix_cpu_timer_create(struct k_itim
    p = current;
   } else {
    p = find_task_by_pid(pid);
-   if (p && p->tgid != current->tgid)
+   if (p && !same_thread_group(p, current))
     p = NULL;
   }
  } else {
@@ -363,7 +363,7 @@ int posix_cpu_timer_create(struct k_itim
    p = current->group_leader;
   } else {
    p = find_task_by_pid(pid);
-   if (p && p->tgid != pid)
+   if (p && !has_group_leader_pid(p))
     p = NULL;
   }
  }
diff --git a/kernel/posix-timers.c b/kernel/posix-timers.c
index 8d2cb05..bc77ba8 100644
--- a/kernel/posix-timers.c
+++ b/kernel/posix-timers.c
@@ -404,7 +404,7 @@ static struct task_struct * good_sigeven

  if ((event->sigev_notify & SIGEV_THREAD_ID ) &&
   (!(rtn = find_task_by_pid(event->sigev_notify_thread_id)) ||
-   rtn->tgid != current->tgid ||
+   !same_thread_group(rtn, current) ||
    (event->sigev_notify & ~SIGEV_THREAD_ID) != SIGEV_SIGNAL))
   return NULL;

@@ -609,7 +609,7 @@ static struct k_itimer * lock_timer(time
   spin_unlock(&idr_lock);

   if ((timr->it_id != timer_id) || !(timr->it_process) ||
-    timr->it_process->tgid != current->tgid) {
+    !same_thread_group(timr->it_process, current)) {
    unlock_timer(timr, *flags);
    timr = NULL;
```

```
   }
diff --git a/kernel/ptrace.c b/kernel/ptrace.c
index 0d7fd07..f90207a 100644
--- a/kernel/ptrace.c
+++ b/kernel/ptrace.c
@@ -169,7 +169,7 @@ int ptrace_attach(struct task_struct *ta
  retval = -EPERM;
  if (task->pid <= 1)
   goto out;
- if (task->tgid == current->tgid)
+ if (same_thread_group(task, current))
   goto out;

 repeat:
diff --git a/kernel/signal.c b/kernel/signal.c
index d6ca591..df785a7 100644
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -1145,7 +1145,7 @@ static int kill_something_info(int sig,

   read_lock(&tasklist_lock);
   for_each_process(p) {
-   if (p->pid > 1 && p->tgid != current->tgid) {
+   if (p->pid > 1 && !same_thread_group(p, current)) {
     int err = group_send_sig_info(sig, info, p);
     ++count;
     if (err != -EPERM)
diff --git a/mm/oom_kill.c b/mm/oom_kill.c
index e276a3c..1a2f687 100644
--- a/mm/oom_kill.c
+++ b/mm/oom_kill.c
@@ -330,7 +330,7 @@ static int oom_kill_task(struct task_str
   * to memory reserves though, otherwise we might deplete all memory.
   */
  do_each_thread(g, q) {
-  if (q->mm == mm && q->tgid != p->tgid)
+  if (q->mm == mm && !same_thread_group(q, p))
    force_sig(SIGKILL, q);
  } while_each_thread(g, q);
```