Subject: [PATCH 14/17] Pid-NS(V3) [PATCH] Allow task_active_pid_ns() to be NULL
Posted by Sukadev Bhattiprolu on Sat, 16 Jun 2007 23:04:30 GMT
View Forum Message <> Reply to Message

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
Subject: [PATCH 14/17] Pid-NS(V3) [PATCH] Allow task_active_pid_ns() to be NULL

In earlier patches we get the pid_t or active pid namespace of a task/process from its 'task_pid()'. But when a process is exiting, the task_pid() can be NULL and we are unable to retrieve the 'pid_t' or 'active pid namespace' of the process.

One way we could try to resolve this is to not drop the reference to task_struct->pid until the task_struct is freed. But that could be an intrusive change and we need to watch for some race conditions.

Rather than try and keep task_pid() valid longer, this patch attempts to maintain the current behavior - i.e callers should expect task_pid() or task_active_pid_ns() to be NULL.

For instance, the pid_nr() function in mainline returns 0 if 'pid' is NULL. Similarly we can have task_active_pid_ns() return NULL pid namespace if 'pid' is NULL and ensure we address the callers on a case-by-case basis as follows:

- is_global_init() - use a global variable to implement this and don't depend on task_active_pid_ns().

- proc_flush_task() called from release_task() - Call proc_flush_task() before detaching 'struct pid' from the task. An alternate implem. could be to get a reference to the pid in release_task() before detaching the pid. Use this reference to find the namespaces the process exists in.

- group_send_siginfo() called from release_task() - In this case the kernel (not user space) sends a signal (SIGCHLD) to parent of process when a thread other than the leader is the last one exiting.

  group_send_siginfo() (in subsequent patch) needs to normally check pid ns of sender to see if they are signalling the child reaper. But since in this case, the signal SIGCHLD is coming from the kernel we already bypass permission checks. We could by pass the pid namespace check also.

P.S: I have reviewed all calls to task_pid(), task_active_pid_ns(), task_child_reaper(), pid_active_upid(), pid_active_pid_ns(), focusing on the callers that pass in a task other than 'current' and they look safe.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
---
 include/linux/pid.h | 8 +++++++-
 include/linux/sched.h | 3 +++
 2 files changed, 10 insertions(+), 1 deletion(-)

Index: lx26-22-rc4-mm2/include/linux/pid.h
==========
--- lx26-22-rc4-mm2.orig/include/linux/pid.h 2007-06-15 18:25:14.000000000 -0700
+++ lx26-22-rc4-mm2/include/linux/pid.h 2007-06-15 18:26:08.000000000 -0700
@@ -144,6 +144,9 @@ extern void zap_pid_ns_processes(struct
  */
 static inline struct upid *pid_active_upid(struct pid *pid)
 {
+ if (!pid)
+ return NULL;
+
  return &pid->upid_list[0];
 }

@@ -152,6 +155,9 @@ static inline struct upid *pid_active_up
  */
 static inline struct pid_namespace *pid_active_pid_ns(struct pid *pid)
 {
+ if (!pid)
+ return NULL;
+
  return pid_active_upid(pid)->pid_ns;
 }

@@ -194,7 +200,7 @@ static inline pid_t pid_to_nr_in_ns(stru
  int i;
  struct upid *upid;

- if (!pid)
+ if (!pid || !ns)
   return 0;

  for (i = 0; i < pid->num_upids; i++) {
Index: lx26-22-rc4-mm2/include/linux/sched.h
==========
--- lx26-22-rc4-mm2.orig/include/linux/sched.h 2007-06-15 18:25:14.000000000 -0700
+++ lx26-22-rc4-mm2/include/linux/sched.h 2007-06-15 18:26:08.000000000 -0700
@@ -1242,6 +1242,9 @@ static inline struct task_struct *task_c
  struct task_struct *reaper;
  struct pid *pid = task_pid(tsk);

```
+ if (!pid)
+  return NULL;
+
  reaper = pid_active_pid_ns(pid)->child_reaper;
  if (tsk == reaper)
    reaper = pid_active_pid_ns_parent(pid)->child_reaper;

--
```

----- End forwarded message -----