
Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by [Sukadev Bhattiprolu](#) on Fri, 03 Aug 2007 21:36:36 GMT
[View Forum Message](#) <> [Reply to Message](#)

Oleg Nesterov [oleg@tv-sign.ru] wrote:

| This is harmless (and note that it is possible that current was actually killed
| with SIGKILL from the parent namespace), but the code imho looks confusing.

| I'd suggest to make zap_pid_ns_processes(void), and start the loop from nr == 1.
| Or zap_pid_ns_processes(struct pid_namespace *pid_ns).

| Oleg.

Agree. Here is the modified patch.

From: Pavel Emelyanov <xemul@openvz.org>

Subject: [PATCH 14/15] Destroy pid namespace on init's death

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>

Terminate all processes in a namespace when the reaper of the namespace
is exiting. We do this by walking the pidmap of the namespace and sending
SIGKILL to all processes.

Changelog:

[Oleg Nesterov]: In zap_pid_ns_processes() wait for any child
rather than iterating over all pid_ts in the pidmap. Clear
TIF_SIGPENDING flag for successive wait() calls.

[Oleg Nesterov]: Ensure the logic works even with multi-threaded
container-init process.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>

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

```
include/linux/pid.h | 1 +
kernel/exit.c      | 27 ++++++
kernel/pid.c       | 38 ++++++
3 files changed, 65 insertions(+), 1 deletion(-)
```

Index: lx26-23-rc1-mm1/include/linux/pid.h

```
=====
--- lx26-23-rc1-mm1.orig/include/linux/pid.h 2007-08-02 11:03:39.000000000 -0700
+++ lx26-23-rc1-mm1/include/linux/pid.h 2007-08-03 13:29:40.000000000 -0700
@@ -118,6 +118,7 @@ extern struct pid *find_ge_pid(int nr, s
```

```

extern struct pid *alloc_pid(struct pid_namespace *ns);
extern void FASTCALL(free_pid(struct pid *pid));
+extern void zap_pid_ns_processes(struct pid_namespace *pid_ns);

/*
 * the helpers to get the pid's id seen from different namespaces
Index: lx26-23-rc1-mm1/kernel/exit.c
=====
--- lx26-23-rc1-mm1.orig/kernel/exit.c 2007-08-02 11:06:36.000000000 -0700
+++ lx26-23-rc1-mm1/kernel/exit.c 2007-08-03 13:56:37.000000000 -0700
@@ -916,7 +916,32 @@ static inline void exit_child_reaper(str
     if (likely(tsk->group_leader != task_child_reaper(tsk)))
         return;

- panic("Attempted to kill init!");
+ if (tsk->nsproxy->pid_ns == &init_pid_ns)
+     panic("Attempted to kill init!");
+
+ /*
+  * @tsk is the last thread in the 'container-init' and is exiting.
+  * Terminate all remaining processes in the namespace and reap them
+  * before exiting @tsk.
+  *
+  * Note that @tsk (last thread of container-init) may not necessarily
+  * be the child-reaper (i.e main thread of container-init) of the
+  * namespace i.e the child_reaper may have already exited.
+  *
+  * Even after a child_reaper exits, we let it inherit orphaned children,
+  * because, pid_ns->child_reaper remains valid as long as there is
+  * at least one living sub-thread in the container init.
+  *
+  * This living sub-thread of the container-init will be notified when
+  * a child inherited by the 'child-reaper' exits (do_notify_parent()
+  * uses __group_send_sig_info()). Further, when reaping child processes,
+  * do_wait() iterates over children of all living sub threads.
+  *
+  * i.e even though 'child_reaper' thread is listed as the parent of the
+  * orphaned children, any living sub-thread in the container-init can
+  * perform the role of the child_reaper.
+  */
+ zap_pid_ns_processes(tsk->nsproxy->pid_ns);
}

fastcall NORET_TYPE void do_exit(long code)
Index: lx26-23-rc1-mm1/kernel/pid.c
=====
--- lx26-23-rc1-mm1.orig/kernel/pid.c 2007-08-02 11:03:39.000000000 -0700

```

```

+++ lx26-23-rc1-mm1/kernel/pid.c 2007-08-03 13:56:12.000000000 -0700
@@ -29,6 +29,7 @@
#include <linux/pid_namespace.h>
#include <linux/init_task.h>
#include <linux/proc_fs.h>
+#include <linux/syscalls.h>

#define pid_hashfn(nr, ns) \
    hash_long((unsigned long)nr + (unsigned long)ns, pidhash_shift)
@@ -593,6 +594,43 @@ out:
    return new_ns;
}

+void zap_pid_ns_processes(struct pid_namespace *pid_ns)
+{
+ int nr;
+ int rc;
+
+ /*
+  * The last thread in the container-init thread group is terminating.
+  * Find remaining pid_ts in the namespace, signal and wait for them
+  * to exit.
+  *
+  * Note: This signals each threads in the namespace - even those that
+  * belong to the same thread group, To avoid this, we would have
+  * to walk the entire tasklist looking a processes in this
+  * namespace, but that could be unnecessarily expensive if the
+  * pid namespace has just a few processes. Or we need to
+  * maintain a tasklist for each pid namespace.
+  */
+ read_lock(&tasklist_lock);
+ nr = next_pidmap(pid_ns, 1);
+ while (nr > 0) {
+ kill_proc_info(SIGKILL, SEND_SIG_PRIV, nr);
+ nr = next_pidmap(pid_ns, nr);
+ }
+ read_unlock(&tasklist_lock);
+
+ do {
+ clear_thread_flag(TIF_SIGPENDING);
+ rc = sys_wait4(-1, NULL, __WALL, NULL);
+ } while (rc != -ECHILD);
+
+ /* Child reaper for the pid namespace is going away */
+ pid_ns->child_reaper = NULL;
+ return;

```

```
+}  
+  
static void do_free_pid_ns(struct work_struct *w)  
{  
    struct pid_namespace *ns, *parent;
```

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>
