Subject: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Pavel Emelianov on Thu, 26 Jul 2007 14:57:39 GMT
View Forum Message <> Reply to Message

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.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
Acked-by: Pavel Emelyanov <xemul@openvz.org>

---

```
 include/linux/pid.h  |    1 +
 include/linux/wait.h |    4 ++++
 kernel/exit.c        |   10 ++++++----
 kernel/pid.c         |   38 ++++++++++++++++++++++++++++++++++++++
 4 files changed, 49 insertions(+), 4 deletions(-)

diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/pid.h linux-2.6.23-rc1-mm1-7/include/linux/pid.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/pid.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/pid.h 2007-07-26 16:36:37.000000000 +0400
@@ -83,6 +92,7 @@ extern void FASTCALL(detach_pid(struct t

 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
diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/wait.h linux-2.6.23-rc1-mm1-7/include/linux/wait.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/wait.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/wait.h 2007-07-26 16:36:36.000000000 +0400
@@ -22,9 +22,13 @@
 #include <linux/list.h>
 #include <linux/stddef.h>
 #include <linux/spinlock.h>
+#include <linux/resource.h>
+#include <asm/siginfo.h>
 #include <asm/system.h>
 #include <asm/current.h>

+long do_wait(pid_t pid, int options, struct siginfo __user *infop,
+  int __user *stat_addr, struct rusage __user *ru);
 typedef struct __wait_queue wait_queue_t;
 typedef int (*wait_queue_func_t)(wait_queue_t *wait, unsigned mode, int sync, void *key);
```

```
 int default_wake_function(wait_queue_t *wait, unsigned mode, int sync, void *key);
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
@@ -895,6 +915,7 @@ fastcall NORET_TYPE void do_exit(long co
 {
  struct task_struct *tsk = current;
  int group_dead;
+ struct pid_namespace *pid_ns = tsk->nsproxy->pid_ns;

  profile_task_exit(tsk);

@@ -905,9 +926,10 @@ fastcall NORET_TYPE void do_exit(long co
  if (unlikely(!tsk->pid))
   panic("Attempted to kill the idle task!");
  if (unlikely(tsk == task_child_reaper(tsk))) {
-  if (task_active_pid_ns(tsk) != &init_pid_ns)
-   task_active_pid_ns(tsk)->child_reaper =
-    init_pid_ns.child_reaper;
+  if (pid_ns != &init_pid_ns) {
+   zap_pid_ns_processes(pid_ns);
+   pid_ns->child_reaper = init_pid_ns.child_reaper;
+  }
   else
    panic("Attempted to kill init!");
  }
@@ -1518,7 +1548,7 @@ static inline int my_ptrace_child(struct
  return (p->parent != p->real_parent);
 }

-static long do_wait(pid_t pid, int options, struct siginfo __user *infop,
+long do_wait(pid_t pid, int options, struct siginfo __user *infop,
       int __user *stat_addr, struct rusage __user *ru)
 {
  DECLARE_WAITQUEUE(wait, current);
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/pid.c linux-2.6.23-rc1-mm1-7/kernel/pid.c
--- linux-2.6.23-rc1-mm1.orig/kernel/pid.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/pid.c 2007-07-26 16:36:37.000000000 +0400
@@ -428,6 +520,44 @@ err_alloc:
  return new_ns;
 }

+void zap_pid_ns_processes(struct pid_namespace *pid_ns)
+{
+ int i;
+ int nr;
+ int nfree;
+ int options = WNOHANG|WEXITED|__WALL;
```

```
+
+repeat:
+ /*
+  * We know pid == 1 is terminating. Find remaining pid_ts
+  * in the namespace, signal them and then wait for them
+  * exit.
+  */
+ nr = next_pidmap(pid_ns, 1);
+ while (nr > 0) {
+  kill_proc_info(SIGKILL, SEND_SIG_PRIV, nr);
+  nr = next_pidmap(pid_ns, nr);
+ }
+
+ nr = next_pidmap(pid_ns, 1);
+ while (nr > 0) {
+  do_wait(nr, options, NULL, NULL, NULL);
+  nr = next_pidmap(pid_ns, nr);
+ }
+
+ nfree = 0;
+ for (i = 0; i < PIDMAP_ENTRIES; i++)
+  nfree += atomic_read(&pid_ns->pidmap[i].nr_free);
+
+ /*
+  * If pidmap has entries for processes other than 0 and 1, retry.
+  */
+ if (nfree < (BITS_PER_PAGE * PIDMAP_ENTRIES - 2))
+  goto repeat;
+
+ return;
+}
+
 static void do_free_pid_ns(struct work_struct *w)
 {
  struct pid_namespace *ns, *parent;
```