Subject: [PATCH 3/3] Signal semantics for pid namespaces
Posted by Sukadev Bhattiprolu on Tue, 11 Sep 2007 04:12:25 GMT
View Forum Message <> Reply to Message

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
Subject: [PATCH 3/3] Signal semantics for pid namespaces

With support for multiple pid namespaces, each pid namespace has a
separate child reaper and this process needs some special handling
of signals.

- The child reaper should appear like a normal process to other
  processes in its ancestor namespaces and so should be killable
  (or not) in the usual way.

    - The child reaper should receive, from processes in it's active
      and decendent namespaces, only those signals for which it has
      installed a signal handler.

- System-wide signals (eg: kill signum -1) from within a child namespace
  should only affect processes within that namespace and descendant
  namespaces. They should not be posted to processes in ancestor or
  sibling namespaces.

- If the sender of a signal does not have a pid_t in the receiver's
  namespace (eg: a process in init_pid_ns sends a signal to a process
  in a descendant namespace), the sender's pid and uid should appear
  as 0 in the signal's 'siginfo' structure.

- Existing rules for SIGIO delivery still apply and a process can
  choose any other process in its namespace and descendant namespaces
  to receive the SIGIO signal.

  The following appears to be incorrect in the fcntl() man page for
  F_SETOWN.

> Sending a signal to the owner process (group) specified by
> F_SETOWN is subject to the same permissions checks as are
> described for kill(2), where the sending process is the one that
> employs F_SETOWN (but see BUGS below).

Current behavior is that the SIGIO signal is delivered on behalf of
the process that caused the event (eg: made data available on the
file) and not the process that called fcntl().

Changelog:
- [Oleg Nesterov]: Used the interfaces, is_current_in_ancestor_pid_ns()
  and is_current_in_same_or_ancestor_pid_ns().

- [Oleg Nesterov]: Clear info.si_uid also when masquerading sender.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
---
kernel/signal.c | 28 ++++++++++++++++++++++++++-
1 file changed, 27 insertions(+), 1 deletion(-)

Index: 2.6.23-rc4-mm1/kernel/signal.c
===
--- 2.6.23-rc4-mm1.orig/kernel/signal.c 2007-09-10 18:42:16.000000000 -0700
+++ 2.6.23-rc4-mm1/kernel/signal.c 2007-09-10 18:42:16.000000000 -0700
@@ -25,6 +25,7 @@
#include <linux/capability.h>
#include <linux/freezer.h>
#include <linux/pid_namespace.h>
+#include <linux/pid.h>
#include <linux/nsproxy.h>
#include <linux/hardirq.h>

@@ -45,7 +46,10 @@ static int sig_init_ignore(struct task_s

// Currently this check is a bit racy with exec(),
// we can _simplify_ de_thread and close the race.
- if (likely(!is_global_init(tsk->group_leader)))
+ if (likely(!is_container_init(tsk->group_leader)))
+ return 0;
+
+ if (is_current_in_ancestor_pid_ns(tsk) && !in_interrupt())
return 0;

return 1;
@@ -681,6 +685,20 @@ static void handle_stop_signal(int sig,
}
}

+static void masquerade_sender(struct task_struct *t, struct sigqueue *q)
+{
+ /*
+ * If the sender does not have a pid_t in the receiver's active
+ * pid namespace, set si_pid to 0 and pretend signal originated
+ * from the kernel.
+ */
+ if (!pid_ns_equal(t)) {
+ q->info.si_pid = 0;
+ q->info.si_uid = 0;
+ q->info.si_code = SI_KERNEL;
+ }
+}

```
+
 static int send_signal(int sig, struct siginfo *info, struct task_struct *t,
    struct sigpending *signals)
 {
@@ -732,6 +750,7 @@ static int send_signal(int sig, struct s
    copy_siginfo(&q->info, info);
    break;
   }
+  masquerade_sender(t, q);
  } else if (!is_si_special(info)) {
   if (sig >= SIGRTMIN && info->si_code != SI_USER)
   /*
@@ -1165,6 +1184,7 @@ EXPORT_SYMBOL_GPL(kill_pid_info_as_uid);
 static int kill_something_info(int sig, struct siginfo *info, int pid)
 {
  int ret;
+
  rcu_read_lock();
  if (!pid) {
   ret = kill_pgrp_info(sig, info, task_pgrp(current));
@@ -1174,6 +1194,12 @@ static int kill_something_info(int sig,

   read_lock(&tasklist_lock);
   for_each_process(p) {
+   /*
+    * System-wide signals only apply to pid namespace
+    * of sender.
+    */
+   if (!is_current_in_same_or_ancestor_pid_ns(p))
+    continue;
    if (p->pid > 1 && !same_thread_group(p, current)) {
     int err = group_send_sig_info(sig, info, p);
     ++count;
```

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers