Subject: [PATCH] Signal semantics for /sbin/init
Posted by Sukadev Bhattiprolu on Sat, 27 Oct 2007 19:00:10 GMT
View Forum Message <> Reply to Message

(This is Oleg's patch with my tweaks to compile, Oleg pls sign-off).

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
Subject: [PATCH 1/3] Signal semantics for /sbin/init

Currently, /sbin/init is protected from unhandled signals by the
"current == child_reaper(current)" check in get_signal_to_deliver().
This is not enough, we have multiple problems:

- this doesn't work for multi-threaded inits, and we can't
  fix this by simply making this check group-wide.

- /sbin/init and kernel threads are not protected from
  handle_stop_signal(). Minor problem, but not good and
  allows to "steal" SIGCONT or change ->signal->flags.

- /sbin/init is not protected from __group_complete_signal(),
  sig_fatal() can set SIGNAL_GROUP_EXIT and block exec(), kill
  sub-threads, set ->group_stop_count, etc.

Also, with support for multiple pid namespaces, we need an ability to
actually kill the sub-namespace's init from the parent namespace. In
this case it is not possible (without painful and intrusive changes)
to make the "should we honor this signal" decision on the receiver's
side.

Hopefully this patch (adds 43 bytes to kernel/signal.o) can solve
these problems.

Notes:

- Blocked signals are never ignored, so init still can receive
  a pending blocked signal after sigprocmask(SIG_UNBLOCK).
  Easy to fix, but probably we can ignore this issue.

- this patch allows us to simplify de_thread() playing games
  with pid_ns->child_reaper.

(Side note: the current behaviour of things like force_sig_info_fault()
 is not very good, init should not ignore these signals and go to the
 endless loop. Exit + panic is imho better, easy to chamge)

Oleg.

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

---
 kernel/signal.c | 44 ++++++++++++++++++++++++++++++++++----------
 1 file changed, 30 insertions(+), 14 deletions(-)

Index: 2.6.23-mm1/kernel/signal.c
===
--- 2.6.23-mm1.orig/kernel/signal.c 2007-10-27 08:48:56.000000000 -0700
+++ 2.6.23-mm1/kernel/signal.c 2007-10-27 08:49:23.000000000 -0700
@@ -26,6 +26,7 @@
 #include <linux/freezer.h>
 #include <linux/pid_namespace.h>
 #include <linux/nsproxy.h>
+#include <linux/hardirq.h>

 #include <asm/param.h>
 #include <asm/uaccess.h>
@@ -39,11 +40,32 @@

 static struct kmem_cache *sigqueue_cachep;

+static int sig_init_ignore(struct task_struct *tsk)
+{

-static int sig_ignored(struct task_struct *t, int sig)
+ // 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)))
+ return 0;
+
+ return 1;
+}
+
+static int sig_task_ignore(struct task_struct *tsk, int sig)
 {
- void __user * handler;
+ void __user * handler = tsk->sighand->action[sig-1].sa.sa_handler;
+
+ if (handler == SIG_IGN)
+ return 1;
+
+ if (handler != SIG_DFL)
+ return 0;

+ return sig_kernel_ignore(sig) || sig_init_ignore(tsk);
+}

```
+
+static int sig_ignored(struct task_struct *t, int sig)
+{
  /*
   * Tracers always want to know about signals..
   */
@@ -58,10 +80,7 @@ static int sig_ignored(struct task_struc
  if (sigismember(&t->blocked, sig))
   return 0;

- /* Is it explicitly or implicitly ignored? */
- handler = t->sighand->action[sig-1].sa.sa_handler;
- return   handler == SIG_IGN ||
-  (handler == SIG_DFL && sig_kernel_ignore(sig));
+ return sig_task_ignore(t, sig);
 }

 /*
@@ -566,6 +585,9 @@ static void handle_stop_signal(int sig,
    */
   return;

+ if (sig_init_ignore(p))
+  return;
+
  if (sig_kernel_stop(sig)) {
   /*
    * This is a stop signal.  Remove SIGCONT from all queues.
@@ -1860,12 +1882,6 @@ relock:
   if (sig_kernel_ignore(signr)) /* Default is nothing. */
    continue;

-  /*
-   * Global init gets no signals it doesn't want.
-   */
-  if (is_global_init(current))
-   continue;
-
   if (sig_kernel_stop(signr)) {
    /*
     * The default action is to stop all threads in
@@ -2317,6 +2333,7 @@ int do_sigaction(int sig, struct k_sigac
  k = &current->sighand->action[sig-1];

  spin_lock_irq(&current->sighand->siglock);
+
  if (oact)
   *oact = *k;
```

```
@@ -2335,8 +2352,7 @@ int do_sigaction(int sig, struct k_sigac
    *   (for example, SIGCHLD), shall cause the pending signal to
    *   be discarded, whether or not it is blocked"
    */
-  if (act->sa.sa_handler == SIG_IGN ||
-      (act->sa.sa_handler == SIG_DFL && sig_kernel_ignore(sig))) {
+  if (sig_task_ignore(current, sig)) {
     struct task_struct *t = current;
     sigemptyset(&mask);
     sigaddset(&mask, sig);
```

_____

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