Subject: Re: [RFC.PATCH] fix /sbin/init signal handling Posted by serue on Tue, 21 Aug 2007 16:05:06 GMT

View Forum Message <> Reply to Message

```
Quoting Oleg Nesterov (oleg@tv-sign.ru):
> (Not for inclusion yet, against 2.6.23-rc2, untested)
>
> 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.
> --- t/kernel/signal.c~INITSIGS 2007-08-19 14:39:35.000000000 +0400
> +++ t/kernel/signal.c 2007-08-19 19:00:27.000000000 +0400
```

```
> @ @ -39,11 +39,35 @ @
> static struct kmem_cache *sigqueue_cachep;
> +static int sig_init_ignore(struct task_struct *tsk)
> +{
> + // Currently this check is a bit racy with exec(),
> + // we can _simplify_ de_thread and close the race.
> + if (likely(!is init(tsk->group leader)))
> + return 0;
> + // ------ Multiple pid namespaces ------
> + // if (current is from tsk's parent pid_ns && !in_interrupt())
> + // return 0;
> + return 1;
> +}
> +static int sig_task_ignore(struct task_struct *tsk, int sig)
> + 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);
> +}
Looks good. AFAICS init gets exactly those signals for which it
installed a signal handler.
>
> static int sig_ignored(struct task_struct *t, int sig)
> {
> - void user * handler;
> /*
   * Tracers always want to know about signals...
> @ @ -58,10 +82,7 @ @ 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);
> }
Looks good.
>
> /*
> @ @ -569,6 +590,9 @ @ 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.
> @ @ -1841,14 +1865,6 @ @ relock:
   if (sig_kernel_ignore(signr)) /* Default is nothing. */
    continue;
>
>
> - * Init of a pid space gets no signals it doesn't want from
> - * within that pid space. It can of course get signals from
   * its parent pid space.
> - */
> - if (current == child reaper(current))
> - continue;
```

Ok, so the idea is that this will now be caught when the signal is sent, using sig_ignored(), (i.e at send_sigqueue, send_group_sigqueue, specific_send_sig_info, and __group_send_sig_info) and so doesn't need to be checked here?

I was hoping that meant that sig init ignore() would always be called with current as the sending process, but I guess that's not the case? At least in get_signal_to_deliver() we might resend a signal, though I guess we assume the signal comes from current->parent, so maybe we can pass that as an argument...

```
if (sig_kernel_stop(signr)) {
>
    * The default action is to stop all threads in
> @ @ -2300,13 +2316,10 @ @ int do sigaction(int sig, struct k sigac
> k = &current->sighand->action[sig-1];
```

```
>
  spin_lock_irq(&current->sighand->siglock);
> - if (signal_pending(current)) {
> - /*
> - * If there might be a fatal signal pending on multiple
> - * threads, make sure we take it before changing the action.
> - */
> + if (current->signal->flags & SIGNAL_GROUP_EXIT) {
 spin unlock irg(&current->sighand->siglock);
> - return -ERESTARTNOINTR;
> + /* The return value doesn't matter, SIGKILL is pending */
> + return -EINTR;
> }
Looks right, based on the original comment.
>
> if (oact)
> @ @ -2327,8 +2340,7 @ @ 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);
Haven't tested, but the patch reads good.
thanks,
-serge
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers
```