

---

Subject: Re: [RFC,PATCH] fix /sbin/init signal handling  
Posted by [Pavel Emelianov](#) on Tue, 21 Aug 2007 09:31:24 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Oleg Nesterov wrote:

> (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 change)  
>  
> 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_cache;
>
> +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);
> +}

```

These two look like the init ignores "less" than a usual task, i.e. the decision of whether a task has to ignore a signal depends on whether the init has and some more. This is... strange :)

```

> 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_struct
> 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);
> }
>
> /*
> @@ -569,6 +590,9 @@ static void handle_stop_signal(int sig,
> */
> return;
>
> + if (sig_init_ignore(p))
> + return;
> +

```

Why do we need for explicit stop handling for init? Shouldn't it be automatically checked in get\_signal\_to\_deliver()?

```

> 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;
> -
> 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->sigband->action[sig-1];
>
> spin_lock_irq(&current->sigband->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_irq(&current->sigband->siglock);
> - return -ERESTARTNOINTR;
> + /* The return value doesn't matter, SIGKILL is pending */

```

```
> + return -EINTR;
> }
>
> 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);
>
>
>
```

---

Containers mailing list

Containers@lists.linux-foundation.org

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

---