

---

Subject: Re: [PATCH 1/3] Signal semantics for /sbin/init  
Posted by [Sukadev Bhattiprolu](#) on Tue, 18 Sep 2007 19:00:52 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Oleg Nesterov [oleg@tv-sign.ru] wrote:

| On 09/13, sukadev@us.ibm.com wrote:

| >

| > Oleg Nesterov [oleg@tv-sign.ru] wrote:

| > | > >

| > | > >> 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.

| > | > >

| > | > > I was wrong. This should be fixed right now. I \_think\_ this is easy,

| > | > > and I was going to finish this patch yesterday, but - sorry! - I just

| > | > > can't switch to "kernel mode" these days, I am fighting with some urgent

| > | > > tasks on my paid job.

| > | > >

| > | > > To respect the current init semantic,

| > |

| > | The current init semantic is broken in many ways ;)

| > |

| > | > shouldn't we discard any unblockable

| > | > signal (STOP and KILL) sent by a process to its pid namespace init process ?

| > |

| > | Yes. And Patch 1/3 (Oleg's patch) in the set I sent, handles this already

| > | (since STOP and KILL are never in the task->blocked list)

| > |

| > |

| > | > Then, all other signals should be handled appropriately by the pid namespace

| > | > init.

| > |

| > |

| > | Yes, I think you are probably right, this should be enough in practice. After all,

| > | only root can send the signal to /sbin/init.

| > |

| > | I agree - the assumption that the container-init will handle these

| > | other signals, simplifies the kernel implementation for now.

| > |

| > |

| > | On my machine, /proc/1/status shows that init doesn't have a handler for

| > | non-ignored SIGUNUSED == 31, though.

| > |

| > | But who knows? The kernel promises some guarantees, it is not good to break them.

| > | Perhaps some strange non-standard environment may suffer.

| > |

| > | > We are assuming that the pid namespace init is not doing anything silly and  
 | > | > I guess it's OK if the consequences are only on the its pid namespace and  
 | > | > not the whole system.  
 | > |  
 | > | The sub-namespace case is very easy afaics, we only need the "signal comes from  
 | > | the parent namespace" check, not a problem if we make the decision on the sender's  
 | > | path, like this patch does.  
 | >  
 | > Yes, patches 2 and 3 of the set already do the ancestor-ns check. no ?  
 |  
 | Yes, I think patches 2-3 are good. But this patch is not. I thought that we  
 | can ignore the "Blocked signals are never ignored" problem, now I am not sure.  
 | It is possible that init temporary blocks a signal which it is not going to  
 | handle.  
 |  
 | Perhaps we can do something like the patch below, but I don't like it. With  
 | this patch, we check the signal handler even if /sbin/init blocks the signal.  
 | This makes the semantics a bit strange for /sbin/init. Hopefully not a problem  
 | in practice, but still not good.

I think this is one step ahead of what we were discussing last week.  
 A container-init that does not have a handler for a fatal signal would  
 survive even if the signal is posted when it is blocked.

|  
 | Unfortunately, I don't know how to make it better. The problem with blocked  
 | signals is that we don't know who is the sender of the signal at the time  
 | when the signal is unblocked.

One solution I was thinking of was to possibly queue pending blocked  
 signals to a container init seperately and then requeue them on the  
 normal queue when signals are unblocked. Its definitely not an easier  
 solution, but might be less intrusive than the "signal from parent ns  
 flag" solution.

i.e suppose we have:

```
struct pid_namespace {
    ...
    struct sigpending cinit_blocked_pending;
    struct sigpending cinit_blocked_shared_pending;
}
```

Signals from ancestor ns are queued as usual on task->pending and  
 task->signal->shared\_pending. They don't need any special handling.

Only signals posted to a container-init from within its namespace  
 need special handling (as in: ignore unhandled fatal signals from

same namespace).

If the container-init has say SIGUSR1 blocked, and a descendant of container-init posts SIGUSR1 to container-init, queue the SIGUSR1 in pid\_namespace->cinit\_blocked\_pending.

When container-init unblocks SIGUSR1, check if there was a pending SIGUSR1 from same namespace (i.e check ->cinit\_blocked\_pending list). If there was and container-init has a handler for SIGUSR1, post SIGUSR1 on task->pending queue and let the container-init handle SIGUSR1.

If there was a SIGUSR1 posted to container init and there is no handler for SIGUSR1, then just ignore the SIGUSR1 (since it would be fatal otherwise).

I chose 'struct pid\_namespace' for the temporary queue, since we need the temporary queues only for container-inits (not for all processes). And having it allocated ahead of time, ensures we can queue the signal even under low-memory conditions.

Just an idea at this point.

|  
| What do you think? Can we live with this oddity? Otherwise, we have to add  
| something like the "the signal is from the parent namespace" flag, and I bet  
| this is not trivial to implement correctly.

I think its reasonable to place some restrictions on container-init processes, so, yes, I think the oddity is fine for now (i.e at least until someone needs a different behavior).

BTW, I ran some tests on this patch and they seem to work as expected :-)  
Will run some more tests today.

|  
| Oleg.  
|  
| --- t/kernel/signal.c~IINITSIGS 2007-08-28 19:15:28.000000000 +0400  
| +++ t/kernel/signal.c 2007-09-17 19:20:24.000000000 +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;
|
| -static int sig_ignored(struct task_struct *t, int sig)
| + // ----- 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;
| + void __user * handler = tsk->sigband->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..
|  */
| @@ -55,13 +79,10 @@ static int sig_ignored(struct task_struct *t, int sig)
|  * signal handler may change by the time it is
|  * unblocked.
|  */
| - if (sigismember(&t->blocked, sig))
| + if (sigismember(&t->blocked, sig) && !sig_init_ignore(t))
|     return 0;
|
| - /* Is it explicitly or implicitly ignored? */
| - handler = t->sigband->action[sig-1].sa.sa_handler;
| - return handler == SIG_IGN ||
| - (handler == SIG_DFL && sig_kernel_ignore(sig));
| + return sig_task_ignore(t, sig);
| }
|
| /*
| @@ -554,6 +575,9 @@ 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.
|      @@ -1822,14 +1846,6 @@ 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)) {
|     if (current->signal->flags & SIGNAL_GROUP_EXIT)
|     continue;
| @@ -2308,8 +2324,7 @@ 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>

---