Subject: [PATCH 1/3] Signal semantics for /sbin/init
Posted by Sukadev Bhattiprolu on Tue, 11 Sep 2007 04:10:30 GMT
View Forum Message <> Reply to Message

(This is Oleg's patch with my pid ns additions. Compiled and unit tested
on 2.6.23-rc4-mm1 with other patches in this set. Oleg pls update this
patch if necessary and sign-off)

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.

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

Index: 2.6.23-rc3-mm1/kernel/signal.c
===================================================================
--- 2.6.23-rc3-mm1.orig/kernel/signal.c 2007-09-05 12:24:32.000000000 -0700
+++ 2.6.23-rc3-mm1/kernel/signal.c 2007-09-05 12:30:12.000000000 -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);
 }

 /*
@@ -568,6 +587,9 @@ 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.
@@ -1862,12 +1884,6 @@ 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
@@ -2319,6 +2335,7 @@ int do_sigaction(int sig, struct k_sigac
  k = &current->sighand->action[sig-1];

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

@@ -2337,8 +2354,7 @@ 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);
```

Subject: Re: [PATCH 1/3] Signal semantics for /sbin/init
Posted by Oleg Nesterov on Tue, 11 Sep 2007 11:19:28 GMT
View Forum Message <> Reply to Message

On 09/10, sukadev@us.ibm.com wrote:
>
> (This is Oleg's patch with my pid ns additions. Compiled and unit tested
> on 2.6.23-rc4-mm1 with other patches in this set. Oleg pls update this
> patch if necessary and sign-off)

Sukadev, my apologies. This patch does need some changes,

> 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.

Please feel free to solve this issue yourself if you wish. As for me, I'm
forgetting about the kernel until at least the next weekend.

Also, Pavel has some ideas how to do this all on receiver's path, perhaps
this makes more sense (not that I completely agree, but I didn't see the
code yet).

Sorry for inconvenience,

Oleg.

Subject: Re: [PATCH 1/3] Signal semantics for /sbin/init
Posted by Cedric Le Goater on Thu, 13 Sep 2007 15:40:27 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 09/10, sukadev@us.ibm.com wrote:
>> (This is Oleg's patch with my pid ns additions. Compiled and unit tested
>> on 2.6.23-rc4-mm1 with other patches in this set. Oleg pls update this
>> patch if necessary and sign-off)
>
> Sukadev, my apologies. This patch does need some changes,
>
>> 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.
>
> Please feel free to solve this issue yourself if you wish. As for me, I'm
> forgetting about the kernel until at least the next weekend.
>
> Also, Pavel has some ideas how to do this all on receiver's path, perhaps
> this makes more sense (not that I completely agree, but I didn't see the
> code yet).

To respect the current init semantic, shouldn't we discard any unblockable
signal (STOP and KILL) sent by a process to its pid namespace init process ?
Then, all other signals should be handled appropriately by the pid namespace
init.

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.

Cheers,

C.

Subject: Re: [PATCH 1/3] Signal semantics for /sbin/init
Posted by Oleg Nesterov on Thu, 13 Sep 2007 16:58:20 GMT
View Forum Message <> Reply to Message

On 09/13, Cedric Le Goater wrote:
>
> Oleg Nesterov wrote:
> > On 09/10, sukadev@us.ibm.com wrote:
> >> (This is Oleg's patch with my pid ns additions. Compiled and unit tested
> >> on 2.6.23-rc4-mm1 with other patches in this set. Oleg pls update this
> >> patch if necessary and sign-off)
> >
> > Sukadev, my apologies. This patch does need some changes,
> >
> >> 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 ?
> 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. 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.

Oleg.

Subject: Re: [PATCH 1/3] Signal semantics for /sbin/init
Posted by Sukadev Bhattiprolu on Fri, 14 Sep 2007 03:00:53 GMT
View Forum Message <> Reply to Message

Oleg Nesterov [oleg@tv-sign.ru] wrote:
| On 09/13, Cedric Le Goater wrote:
| >
| > Oleg Nesterov wrote:
| > > On 09/10, sukadev@us.ibm.com wrote:
| > >> (This is Oleg's patch with my pid ns additions. Compiled and unit tested
| > >> on 2.6.23-rc4-mm1 with other patches in this set. Oleg pls update this
| > >> patch if necessary and sign-off)
| > >
| > > Sukadev, my apologies. This patch does need some changes,
| > >
| > >> 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 ?

|
| Oleg.

Subject: Re: Re: [PATCH 1/3] Signal semantics for /sbin/init
Posted by Daniel Pittman on Fri, 14 Sep 2007 10:16:29 GMT
View Forum Message <> Reply to Message

Oleg Nesterov <oleg@tv-sign.ru> writes:
> On 09/13, Cedric Le Goater wrote:
>> Oleg Nesterov wrote:

[...]

>> To respect the current init semantic,
>
> The current init semantic is broken in many ways ;)

Yup. They sure are, but they are pretty set in stone by now. :)

>> shouldn't we discard any unblockable signal (STOP and KILL) sent by a
>> process to its pid namespace init process ? 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. 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.

In this case "strange non-standard environments" would mean anyone
running the 'upstart' daemon from recent Ubuntu -- it depends on the
current kernel semantics.

Regards,
Daniel
--
Daniel Pittman <daniel@cybersource.com.au> Phone: 03 9621 2377
Level 4, 10 Queen St, Melbourne Web: http://www.cyber.com.au
Cybersource: Australia's Leading Linux and Open Source Solutions Company

---

Subject: Re: [PATCH 1/3] Signal semantics for /sbin/init
Posted by Oleg Nesterov on Mon, 17 Sep 2007 15:21:14 GMT
View Forum Message <> Reply to Message

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.

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.

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.

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_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;

-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->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..
   */
@@ -55,13 +79,10 @@ static int sig_ignored(struct task_struc
   * 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->sighand->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

---

Subject: Re: Re: [PATCH 1/3] Signal semantics for /sbin/init
Posted by Oleg Nesterov on Mon, 17 Sep 2007 15:22:23 GMT
View Forum Message <> Reply to Message

On 09/14, Daniel Pittman wrote:
>
> Oleg Nesterov <oleg@tv-sign.ru> writes:
> > On 09/13, Cedric Le Goater wrote:
> >> Oleg Nesterov wrote:
>
> [...]
>
> >> To respect the current init semantic,
> >
> > The current init semantic is broken in many ways ;)
>
> Yup. They sure are, but they are pretty set in stone by now. :)
>
> >> shouldn't we discard any unblockable signal (STOP and KILL) sent by a
> >> process to its pid namespace init process ? 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. 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.
>

> In this case "strange non-standard environments" would mean anyone
> running the 'upstart' daemon from recent Ubuntu -- it depends on the
> current kernel semantics.

Just curious, could you tell more? What "current kernel semantics" do you mean?

Do you mean that the 'upstart' daemon sends the unhandled signal to init?

Oleg.

Subject: Re: Re: [PATCH 1/3] Signal semantics for /sbin/init
Posted by Daniel Pittman on Mon, 17 Sep 2007 23:20:34 GMT
View Forum Message <> Reply to Message

Oleg Nesterov <oleg@tv-sign.ru> writes:
> On 09/14, Daniel Pittman wrote:
>> Oleg Nesterov <oleg@tv-sign.ru> writes:
>> > On 09/13, Cedric Le Goater wrote:
>> >> Oleg Nesterov wrote:
>>
>> [...]
>>
>> >> To respect the current init semantic,
>> >
>> > The current init semantic is broken in many ways ;)
>>
>> Yup. They sure are, but they are pretty set in stone by now. :)
>>
>> >> shouldn't we discard any unblockable signal (STOP and KILL) sent by a
>> >> process to its pid namespace init process ? 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. 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.
>>
>> In this case "strange non-standard environments" would mean anyone

>> running the 'upstart' daemon from recent Ubuntu -- it depends on the
>> current kernel semantics.
>
> Just curious, could you tell more? What "current kernel semantics" do
> you mean?

The semantics where (inside the container) init is protected from a wide range of unhandled signals by default.

> Do you mean that the 'upstart' daemon sends the unhandled signal to
> init?

Well, yes and no: upstart does not install a handler for several signals that the traditional sysvinit package uses -- notably, USR1 and USR2 which can trigger reloading of inittab.

The Debian/Ubuntu init scripts still send that signal to the init process during boot to ensure compatibility with the traditional init package.

The current kernel semantics ensure that upstart can do nothing and the unhandled signal does it no harm -- expect, under OpenVZ while getting Ubuntu working I found that it was not protected and the init script would simply kill upstart dead.

The upstream developers of upstart feel that the containers should provide the same semantics as a raw init, and given that an unknown number of end users will have their own administration systems that depend on the same assumptions about how init works I tend to agree.

So, upstart never sends itself a signal that it can't handle, but the rest of the OS environment can.

Regards,
Daniel

(I tend to think the default protection was a mistake, too, but historic mistakes are todays standards. :/)
--
Daniel Pittman <daniel@cybersource.com.au> Phone: 03 9621 2377
Level 4, 10 Queen St, Melbourne Web: http://www.cyber.com.au
Cybersource: Australia's Leading Linux and Open Source Solutions Company
_____

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 containier 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_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;

```
|
| -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->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..
|    */
| @@ -55,13 +79,10 @@ static int sig_ignored(struct task_struc
|    * 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->sighand->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);
```

Subject: Re: [PATCH 1/3] Signal semantics for /sbin/init
Posted by Sukadev Bhattiprolu on Thu, 27 Sep 2007 03:04:53 GMT
View Forum Message <> Reply to Message

Oleg,

Any thoughts on how to proceed with this patchset ? While not complete with respect to blocked signals and container init, would this patchset make semantics slightly better than they are today (container-init can be terminated from within the container) ?

Suka

sukadev@us.ibm.com [sukadev@us.ibm.com] wrote:
| 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 containier 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_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;

```
| |
| | -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->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..
| |    */
| | @@ -55,13 +79,10 @@ static int sig_ignored(struct task_struc
| |    * 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->sighand->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

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

Subject: Re: [PATCH 1/3] Signal semantics for /sbin/init
Posted by Oleg Nesterov on Fri, 28 Sep 2007 10:36:15 GMT
View Forum Message <> Reply to Message

Hi Sukadev,

On 09/26, sukadev@us.ibm.com wrote:

>
> Any thoughts on how to proceed with this patchset ? While not complete
> with respect to blocked signals and container init, would this patchset
> make semantics slightly better than they are today (container-init can
> be terminated from within the container) ?

Please do what you think right, and please use/drop this patch as you wish.

I am really sorry, I am very busy with my paid job, and it is very far from
the kernel hacking. I don't have the time/energy to follow :(

Besides, I've already lost the track on what's going on in this area...
Perhaps I will be able to return in a couple of weeks, but can't promise
anything.

Good luck ;)

Oleg.

Subject: Re: [PATCH 1/3] Signal semantics for /sbin/init
Posted by serue on Mon, 01 Oct 2007 17:00:35 GMT
View Forum Message <> Reply to Message

Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):
> 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 personally prefer the flag solution just because it will remain
clear why it is there, whereas understanding why the separate queue
is there will be harder unless it is named something like
"child_contaienr_init_blocked_pending_queue".

But still it may be the way to go. Have you coded up a version of this?

thanks,
-serge

> 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 containier 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_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;
> |
> | -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->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..
> |    */
> | @@ -55,13 +79,10 @@ static int sig_ignored(struct task_struc
> |    * 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->sighand->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

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

---

Subject: Re: [PATCH 1/3] Signal semantics for /sbin/init
Posted by Sukadev Bhattiprolu on Mon, 01 Oct 2007 17:47:20 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn [serue@us.ibm.com] wrote:
| Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):
| > 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 personally prefer the flag solution just because it will remain
| clear why it is there, whereas understanding why the separate queue
| is there will be harder unless it is named something like
| "child_contaienr_init_blocked_pending_queue".

In my first version of the "flag" solution, I stored the flag in the
sigqueue structure. The problem with that approach was that if the
allocation of the sigqueue failed, we would not know if the sender
was in parent ns. Note that we post a signal to the process (add to
signals->signal set) even if this allocation fails.

By storing the signal info in 'struct pid_namespace' we avoid having
to allocate when posting the signal.

I agree that a descriptive name is needed. but since the fields are
in 'struct pid_namespace' I was thinking 'child' was not necessary.
Maybe 'cinit' instead of 'container init'. Also 'pending' somehow
implies a 'queue' - no ?

| But still it may be the way to go. Have you coded up a version of this?

I was playing with a slightly different solution that I could modify
for this. But I can code that up in a couple of days. Just wanted to
see if it was interesting approach at all.

|
| thanks,
| -serge

|
| > 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 containier 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_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;
| > |
| > | -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->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..
| > |    */
| > | @@ -55,13 +79,10 @@ static int sig_ignored(struct task_struc
| > |    * 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->sighand->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

_____

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

---

Subject: Re: [PATCH 1/3] Signal semantics for /sbin/init
Posted by serue on Mon, 01 Oct 2007 18:08:49 GMT
View Forum Message <> Reply to Message

Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):
> Serge E. Hallyn [serue@us.ibm.com] wrote:
> | Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):
> | > 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 personally prefer the flag solution just because it will remain
> | clear why it is there, whereas understanding why the separate queue

> | is there will be harder unless it is named something like
> | "child_contaienr_init_blocked_pending_queue".
>
> In my first version of the "flag" solution, I stored the flag in the
> sigqueue structure. The problem with that approach was that if the
> allocation of the sigqueue failed, we would not know if the sender
> was in parent ns. Note that we post a signal to the process (add to
> signals->signal set) even if this allocation fails.
>
> By storing the signal info in 'struct pid_namespace' we avoid having
> to allocate when posting the signal.
>
> I agree that a descriptive name is needed. but since the fields are
> in 'struct pid_namespace' I was thinking 'child' was not necessary.
> Maybe 'cinit' instead of 'container init'. Also 'pending' somehow
> implies a 'queue' - no ?

Sure. So cinit_blocked_pending? Sorry, I see now that that was what
you had :)

Please send a patch when you can. It sounds promising.

> | But still it may be the way to go. Have you coded up a version of this?
>
> I was playing with a slightly different solution that I could modify
> for this. But I can code that up in a couple of days. Just wanted to
> see if it was interesting approach at all.

There is interest in getting this issue solved :)

thanks,
-serge

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

---

Subject: Re: [PATCH 1/3] Signal semantics for /sbin/init
Posted by Sukadev Bhattiprolu on Fri, 05 Oct 2007 04:30:30 GMT
View Forum Message <> Reply to Message

| > | > 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.

Hmm. Stumbled upon an issue while coding this up.

For real time signals, is the order in which signals are delivered
important ? If so, the above solution won't work, because when requeing
signals from pid namespace to the task, we may lose the order of signals.

i.e:
 - signal sig1 is blocked.

 - sig1 is posted and queued on the ns->cinit_blocked_pending queue

 - just as we enter sigprocmask() to unblock sig1, if sig2 is
   posted, we queue sig2 on task->pending queue since sig2 is not
   blocked.

 - when we actually unblock sig1, if we requeue ns->cinit_blocked_pending
   signals (i.e sig1) after sig2 we end up reversing the order of the
   signals sig1 and sig2.

   if we requeue the ns-> signals before sig2 and another sig1
   was received after the sig2 and just before the unblock, we
   again lose the order.

The order of signals is not important to legacy signals (< SIGRTMIN) but is
probably an issue for signals SIGRTMIN..SIGRTMAX.

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

---

Subject: Re: [PATCH 1/3] Signal semantics for /sbin/init
Posted by serue on Mon, 08 Oct 2007 14:36:49 GMT
View Forum Message <> Reply to Message

Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):
> | > | > 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.
>
> Hmm. Stumbled upon an issue while coding this up.
>
> For real time signals, is the order in which signals are delivered
> important ?

A very quick, inadequate google search suggests that while order is

important, the order in which they should be delivered is in increasing
signal number. So that should be easy enough to maintain with this
type of patch, right?

-serge

> If so, the above solution won't work, because when requeing
> signals from pid namespace to the task, we may lose the order of signals.
>
> i.e:
> - signal sig1 is blocked.
>
> - sig1 is posted and queued on the ns->cinit_blocked_pending queue
>
> - just as we enter sigprocmask() to unblock sig1, if sig2 is
>   posted, we queue sig2 on task->pending queue since sig2 is not
>   blocked.
>
> - when we actually unblock sig1, if we requeue ns->cinit_blocked_pending
>   signals (i.e sig1) after sig2 we end up reversing the order of the
>   signals sig1 and sig2.
>
>   if we requeue the ns-> signals before sig2 and another sig1
>   was received after the sig2 and just before the unblock, we
>   again lose the order.
>
> The order of signals is not important to legacy signals (< SIGRTMIN) but is
> probably an issue for signals SIGRTMIN..SIGRTMAX.

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

---

Subject: Re: [PATCH 1/3] Signal semantics for /sbin/init
Posted by Sukadev Bhattiprolu on Mon, 08 Oct 2007 15:42:00 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn [serue@us.ibm.com] wrote:
| Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):
| > | > | > 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.
| >
| > Hmm. Stumbled upon an issue while coding this up.
| >

| > For real time signals, is the order in which signals are delivered
| > important ?
|
| A very quick, inadequate google search suggests that while order is
| important, the order in which they should be delivered is in increasing
| signal number. So that should be easy enough to maintain with this
| type of patch, right?

Yes. I could stick all the signals to the end of the queue and the
existing code should pick them up in proper order.

Thanks,

Suka

|
| -serge
|
| > If so, the above solution won't work, because when requeing
| > signals from pid namespace to the task, we may lose the order of signals.
| >
| > i.e:
| > - signal sig1 is blocked.
| >
| > - sig1 is posted and queued on the ns->cinit_blocked_pending queue
| >
| > - just as we enter sigprocmask() to unblock sig1, if sig2 is
| >   posted, we queue sig2 on task->pending queue since sig2 is not
| >   blocked.
| >
| > - when we actually unblock sig1, if we requeue ns->cinit_blocked_pending
| >   signals (i.e sig1) after sig2 we end up reversing the order of the
| >   signals sig1 and sig2.
| >
| >   if we requeue the ns-> signals before sig2 and another sig1
| >   was received after the sig2 and just before the unblock, we
| >   again lose the order.
| >
| > The order of signals is not important to legacy signals (< SIGRTMIN) but is
| > probably an issue for signals SIGRTMIN..SIGRTMAX.