Subject: Re: [PATCH] Signal semantics for /sbin/init
Posted by Sukadev Bhattiprolu on Tue, 30 Oct 2007 21:37:10 GMT
View Forum Message <> Reply to Message

```
| sukadev@us.ibm.com writes:
|
| > (This is Oleg's patch with my tweaks to compile, Oleg pls sign-off).
| > ---
| >
| > From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
| > Subject: [PATCH 1/3] Signal semantics for /sbin/init
| >
| > Currently, /sbin/init is protected from unhandled signals by the
| > "current == child_reaper(current)" check in get_signal_to_deliver().
| > This is not enough, we have multiple problems:
| >
| >  - this doesn't work for multi-threaded inits, and we can't
| >    fix this by simply making this check group-wide.
| >
| >  - /sbin/init and kernel threads are not protected from
| >    handle_stop_signal(). Minor problem, but not good and
| >    allows to "steal" SIGCONT or change ->signal->flags.
| >
| >  - /sbin/init is not protected from __group_complete_signal(),
| >    sig_fatal() can set SIGNAL_GROUP_EXIT and block exec(), kill
| >    sub-threads, set ->group_stop_count, etc.
| >
| > Also, with support for multiple pid namespaces, we need an ability to
| > actually kill the sub-namespace's init from the parent namespace. In
| > this case it is not possible (without painful and intrusive changes)
| > to make the "should we honor this signal" decision on the receiver's
| > side.
| >
| > Hopefully this patch (adds 43 bytes to kernel/signal.o) can solve
| > these problems.
| >
| > Notes:
| >
| >  - Blocked signals are never ignored, so init still can receive
| >    a pending blocked signal after sigprocmask(SIG_UNBLOCK).
| >    Easy to fix, but probably we can ignore this issue.
| >
| >  - this patch allows us to simplify de_thread() playing games
| >    with pid_ns->child_reaper.
| >
| > (Side note: the current behaviour of things like force_sig_info_fault()
| >  is not very good, init should not ignore these signals and go to the
| >  endless loop. Exit + panic is imho better, easy to chamge)
```

| >
| > Oleg.
| >
| > Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
|
| I kept thinking about the problem and the ignored signal issue
| really disturbed me, because we did not have a clean definition
| on how things are supposed to work. I do think Oleg was on the right
| track.
|
| If we make the rule:
| When sending a signal to init. The presence of a signal handler
| that is neither SIG_IGN nor SIG_DFL allows the signal to be sent to
| init. If the signal is not sent it is silently dropped, without
| becoming pending. Further if init specifies it's signal handler as
| SIG_IGN or SIG_DFL all pending signals will be dropped.
|
| The only noticeable user space difference from todays init
| is that it no longer needs to worry about signals becoming
| pending when it has them marked as SIG_DFL or SIG_IGN, and
| then it blocks them.

Yes, thats a small change in semantics to blocked signals and
may not matter much in practice.

|
| This change by making the presence of a signal handler effectively
| a permission check allows us to do all of the work before
| we enqueue the signal. Which means that we can now allow
| force_sig_info to send signals to init and that panic the kernel
| instead of going into an infinite busy loop taking an exception
| sending a signal and then retaking the same exception.
|
| This change also makes it possible to easily implement the
| the desired semantics of /sbin/init for pid namespaces where
| outer processes can kill init but processes inside the pid
| namespace can not.
|
| Please take a look and tell me what you think.

Overall, it looks good, couple of questions below. Will port my
patches and test it out.

|
| diff --git a/kernel/signal.c b/kernel/signal.c
| index 4537bdd..79856eb 100644
| --- a/kernel/signal.c
| +++ b/kernel/signal.c

```
| @@ -546,6 +546,22 @@ static int check_kill_permission(int sig, struct siginfo *info,
|   return security_task_kill(t, info, sig, 0);
|  }
|
| +static int sig_available(struct task_struct *tsk, int sig)
```

Hmm. IMHO, 'available' is not immediately obvious/clear.

```
| +{
| + void __user * handler;
| + int available = 1;
| +
| + if (likely(!is_global_init(tsk)))
| +  goto out;
```

With multiple pid namespaces, I guess, this would become

```
 if (!is_container_init(tsk))
  goto out;

 /* from parent namespace, don't ignore */
 if (!task_in_descendant_ns(tsk))
  goto out;
```

If this is correct, I have a question below re: do_sigaction.

```
| +
| + handler = tsk->sighand->action[sig-1].sa.sa_handler;
| + available = (handler != SIG_IGN) &&
| +      (handler != SIG_DFL);
```

Can we use sig_user_defined() for the above checks ? sig_available()
looks like an extension of sig_user_defined() for init. How about
sig_init_user_defined() or reverse the logic and use sig_init_ignore()
like Oleg did ?

```
| +out:
| + return available;
| +}
| +
| +
|  /* forward decl */
|  static void do_notify_parent_cldstop(struct task_struct *tsk, int why);
|
| @@ -948,6 +964,9 @@ __group_send_sig_info(int sig, struct siginfo *info, struct task_struct *p)
|   int ret = 0;
|
|   assert_spin_locked(&p->sighand->siglock);
| + if (!sig_available(p, sig))
| +  return ret;
| +

|   handle_stop_signal(sig, p);
|
|   /* Short-circuit ignored signals.  */
| @@ -1379,6 +1398,11 @@ send_group_sigqueue(int sig, struct sigqueue *q, struct task_struct
*p)
|   read_lock(&tasklist_lock);
|   /* Since it_lock is held, p->sighand cannot be NULL. */
|   spin_lock_irqsave(&p->sighand->siglock, flags);
| + if (!sig_available(p, sig)) {
| +  ret = 1;
| +  goto out;
| + }
| +
|   handle_stop_signal(sig, p);
|
|   /* Short-circuit ignored signals.  */
```

Hmm. I see now that Oleg's approach would result in the sig_init_ignore()
check being done twice (once in handle_stop_signal() and once in the following
sig_ignored()).

Is that why you don't fold sig_available() into sig_ignored() ? Or, there
other more important/correctness issues as well ?

```
| @@ -1860,12 +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
| @@ -2246,8 +2264,10 @@ static int do_tkill(int tgid, int pid, int sig)
|     */
|    if (!error && sig && p->sighand) {
|     spin_lock_irq(&p->sighand->siglock);
| -   handle_stop_signal(sig, p);
| -   error = specific_send_sig_info(sig, &info, p);
| +   if (sig_available(p, sig)) {
| +    handle_stop_signal(sig, p);
| +    error = specific_send_sig_info(sig, &info, p);
| +   }
|     spin_unlock_irq(&p->sighand->siglock);
|    }
|   }
| @@ -2336,7 +2356,8 @@ int do_sigaction(int sig, struct k_sigaction *act, struct k_sigaction
*oact)
|     *   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))) {
| +     (act->sa.sa_handler == SIG_DFL && sig_kernel_ignore(sig)) ||
| +     !sig_available(current, sig)) {
|     struct task_struct *t = current;
|     sigemptyset(&mask);
|     sigaddset(&mask, sig);
```

Not sure about this. Consider that we extend the sig_available() as
I mentioned above for container_init(). Then suppose following sequence
occurs:

- container-init receives a fatal signal say SIGUSR1, from parent-ns
  i.e the signal is pending.

- before processing the pending signal, the container-init sets
  the handler to SIG_DFL (which is to terminate).

  Will we then discard the pending SIGUSR1 even though it was from
  from parent ns ?