Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Oleg Nesterov on Wed, 01 Aug 2007 19:48:11 GMT
View Forum Message <> Reply to Message

On 07/31, sukadev@us.ibm.com wrote:

```
>
> Oleg Nesterov [oleg@tv-sign.ru] wrote:
> | >
> | > @@ -925,9 +926,10 @@ fastcall NORET_TYPE void do_exit(long co
> | >   if (unlikely(!tsk->pid))
> | >    panic("Attempted to kill the idle task!");
> | >   if (unlikely(tsk == task_child_reaper(tsk))) {
> | > -  if (task_active_pid_ns(tsk) != &init_pid_ns)
> | > -   task_active_pid_ns(tsk)->child_reaper =
> | > -     init_pid_ns.child_reaper;
> | > +  if (pid_ns != &init_pid_ns) {
> | > +   zap_pid_ns_processes(pid_ns);
> | > +   pid_ns->child_reaper = init_pid_ns.child_reaper;
> | > +  }
> | >    else
> | >     panic("Attempted to kill init!");
> | >   }
> |
> | Just to remind you, this is not right when init is multi-threaded,
> | we should do this only when the last thread exits.
>
> Sorry, I needed to clarify somethings about the multi-threaded init. I
> got the impresssion that you were sending a patch for the existing bug,
> and meant to review/clarify in the context of the patch.
```

Ah, sorry, I forgot to send the patch to fix the bug in mainline.
Will try to do tomorrow, please feel free to do this if you wish.

> Our current definition of is_container_init() and task_child_reaper()
> refer only to the main-thread of the container-init (since they check
> for pid_t == 1)

Yes.

> If the main-thread is exiting and is the last thread in the group,
> we want terminate other processes in the pid ns (simple case).

Yes.

> If the main thread is exiting, but is not the last thread in the
> group, should we let it exit and let the next thread in the group
> the reaper of the pid ns ?

We can, but why? The main thread's task_struct can't go away until all
sub-threads exit. Its ->nsproxy will be NULL, but this doesn't matter.

```
>  Then we would have the pid ns w/o a container-init (i.e reaper
>  does not have a pid_t == 1, but probably does not matter).
>
>  And, when this last thread is exiting, we want to terminate other
>  processes in the ns right ?
```

Yes, when this last thread is exiting, the entire process is exiting.

```
> | > +void zap_pid_ns_processes(struct pid_namespace *pid_ns)
> | > +{
> | > + int nr;
> | > + int rc;
> | > + int options = WEXITED|__WALL;
> | > +
> | > + /*
> | > +  * We know pid == 1 is terminating. Find remaining pid_ts
> | > +  * in the namespace, signal them and then wait for them
> | > +  * exit.
> | > +  */
> | > + nr = next_pidmap(pid_ns, 1);
> | > + while (nr > 0) {
> | > +  kill_proc_info(SIGKILL, SEND_SIG_PRIV, nr);
> | > +  nr = next_pidmap(pid_ns, nr);
> | > + }
> |
> | Without tasklist_lock held this is not reliable.
>
> Ok. BTW, find_ge_pid() also walks the pidmap, but does not seem to hold
> the tasklist_lock. Is that bc its only used in /proc ?
```

Yes, but this is something different. With or without tasklist_lock,
find_ge_pid()/next_tgid() is not "reliable" (note that alloc_pid() doesn't
take tasklist), but this doesn't matter for /proc.

We should take tasklist_lock to prevent the new process creation.
We can have the "false positives" (copy_process() in progress, PGID/SID
pids), but this is OK. Note that copy_process() checks signal_pending()
after write_lock_irq(&tasklist_lock), that is why it helps.

Oleg.