Subject: Re: [PATCH 1/2] kill_something_info: misc cleanups
Posted by ebiederm on Sun, 17 Dec 2006 01:09:44 GMT
View Forum Message <> Reply to Message

I'm CC'ing the containers list these namespaces are what it is setup
to talk about.

Oleg Nesterov <oleg@tv-sign.ru> writes:

```
> On 12/16, Eric W. Biederman wrote:
>>
>> Oleg Nesterov <oleg@tv-sign.ru> writes:
>>
>> > On top of
>> >  signal-rewrite-kill_something_info-so-it-uses-newer-helpers.patch
>> >
>> > - Factor out sending PIDTYPE_PGID wide signals.
>> >
>> > - Use is_init(p) instead of "p->pid > 1". We don't hash idle threads
> anymore,
>> >   no need to worry about p->pid == 0.
>>
>>
>> I do not believe is_init is the proper function here.
>
> Ok. How about child_reaper() for now? "p->pid == 1" doesn't look
> good either.
```

Ok. It is probably reasonable to introduce a helper at this point so
we can be clear on what is happening. I called it pspace_leader
in my proof of concept implementation. pid_leader is probably more
concise and more in line with the rest of the naming though.

```
>>                                                   In the presence
>> of multiple pid namespaces
>
> In that case we should use something else than for_each_process() to filter
> out task from different namespaces, no?
```

Probably I haven't quite gotten that far yet. In my proof of concept
implementation I simply did more filtering on the task list. i.e. I
did this:

```
> int __kill_pspace_info(int sig, struct siginfo *info, struct pspace *pspace)
> {
>         struct task_struct *p = NULL;
>         int retval = 0, count = 0;
>
>	for_each_process(p) {
>		int err;
>		/* Skip the current pspace leader */
>		if (current_pspace_leader(p))
>			continue;
>
>		/* Skip the sender of the signal */
>		if (p->signal == current->signal)
>			continue;
>
>		/* Skip processes outside the target process space */
>		if (!in_pspace(pspace, p))
>			continue;
>
>		/* Finally it is a good process send the signal. */
>		err = group_send_sig_info(sig, info, p);
>		++count;
>		if (err != -EPERM)
>			retval = err;
>	}
>	return count ? retval : -ESRCH;
> }
>
```

The difficult part is the recursive nature of the pid namespace. We
can't really keep a separate process list for each. So if we wanted
an efficient traversal the only thing I can see is to put the pids
into a radix tree/trie and walk that.

I haven't gotten to this part yet in the production kernel. I am
close but not quite there yet.

```
>>                              the intention is for is_init to catch all of
>> the special handling (except signal behavior) for the init process.
>>
>> That way when we have multiple processes with pid == 1 we know which
>> one we care about.
>
> I must admit, I don't understand what is the purpose of pid namespace.
> The current implementation looks incomplete. For example, mk_pid()
> takes pid_namespace into account, but find_pid() (and thus attach_pid())
> does not. Shouldn't pid_hash[] live in the "struct pid_namespace" ?
```

It isn't yet. Just a bit of scaffolding to make it possible. However must
of the support code is in place. As I see it the first step is getting
everything using struct pid, pid_nr, and getting rid of must uses of
daemonize() and kernel_thread.

Once I have that actually implementing the pid namespace in a usable form
will be a couple of simple patches.

What I am looking at for basic semantics is that each process in
addition to have a unique pid in it's own pid namespace will have a
unique pid in each pid namespace up to the root of the process tree.
I don't expect this to need to be more than about 3 pids for the first
implementation and then CLONE_NEW_PID_NAMESPACE and then return an out
of resources error.

The pid namespace is fundamentally recursive because otherwise things
like wait and kill (basic process control) won't work if they don't
have pids to work with. We need to allow for migration which means
pids can be assigned on another machine which means we have to work
with arbitrary pid values.

I have gotten the worst of them but there are still quite a few pid_t
references that need to be fixed up. The two that stand out in my
head are pids in the siginfo of signal sending and credentials
on unix domain sockets.

When it is done from inside of the pid namespace you will have your
own separate set of pids. Essentially what we are building is chroot
on steroids. Separate instances of the pid namespace, the ipc
namespace, and the mount namespace pretty much will result in the
feeling of having the machine to yourself (when you are inside them).

Eric