Subject: Re: [PATCH 2/2] pidns: Support unsharing the pid namespace.
Posted by Rob Landley on Thu, 24 Feb 2011 01:12:46 GMT
View Forum Message <> Reply to Message

On 02/15/2011 10:53 AM, Daniel Lezcano wrote:
> From: Eric W. Biederman <ebiederm@xmission.com>
>
> - Allow CLONEW_NEWPID into unshare. - Pass both nsproxy->pid_ns and
> task_active_pid_ns to copy_pid_ns As they can now be different.
>
> Unsharing of the pid namespace unlike unsharing of other namespaces
> does not take effect immediately. Instead it affects the children
> created with fork and clone. The first of these children becomes the
> init process of the new pid namespace, the rest become oddball
> children of pid 0. From the point of view of the new pid namespace
> the process that created it is pid 0, as it's pid does not map.
>
> A couple of different semantics were considered but this one was
> settled on because it is easy to implement and it is usable from pam
> modules. The core reasons for the existence of unshare.

Hmmm...

The userspace semantics I expected were for unshare(CLONE_NEWPID) to:

A) make the current process be PID 1 in the new namespace.

B) reparent_to_init() any existing children as if the process that
called unshare() had exited. (Because those children are not in the new
PID namespace.)

Is there a reason to implement it in some way other than that?

```
 if (!fork) {
   unshare(CLONE_NEWUSER);
   exec();
 }
```

> I took a survey of the callers of pam modules and the following
> appears to be a representative sample of their logic. { setup stuff
> include pam child = fork(); if (!child) { setuid() exec /bin/bash }
> waitpid(child);
>
> pam and other cleanup }

And calling unshare() right before calling setuid() seems the logical
thing to do there...?

Currently unshare() works like chroot(). You're making it act like vfork() where the process that did this is in a strange halfway state until it creates new children at which point magic happens. I don't understand why this is an improvement, especially since none of the other flags you can feed to unshare do that.

> As you can see there is a fork to create the unprivileged user space
> process. Which means that the unprivileged user space process will
> appear as pid 1 in the new pid namespace.

Meaning the process that called unshare() becomes the idle task? Or this process isn't actually in the new PID namespace?

> Further most login processes do not cope with extraneous children
> which means shifting the duty of reaping extraneous child process to
> the creator of those extraneous children makes the system more
> comprehensible.

We already have reparent_to_init() happening when a process dies. That can't be adapted/reused?

> The practical reason for this set of pid namespace semantics is that
> it is simple to implement and verify they work correctly. Whereas an
> implementation that requires changing the struct pid on a process
> comes with a lot more races and pain. Not the least of which is that
> glibc caches getpid().

So unshare() in the libc needs to flush that cache. Presumably a one line patch.

> These semantics are implemented by having two notions of the pid
> namespace of a process. There is task_active_pid_ns which is the pid
> namspace the process was created with and the pid namespace that all
> pids are presented to that process in. The task_active_pid_ns is
> stored in the struct pid of the task.

Having two PID namespaces for each process is the simple answer?

> There is the pid namespace that will be used for children that pid
> namespace is stored in task->nsproxy->pid_ns.
>
> There is one really nasty corner case in all of this. Which pid
> namespace are you in if your parent unshared it's pid namespace and
> then on clone you also unshare the pid namespace. To me there are
> only two possible answers. Either the cases is so bizarre and we
> deny it completely. or the new pid namespace is a descendent of our
> parent's active pid namespace, and we ignore the
> task->nsproxy->pid_ns.

>
> To that end I have modified copy_pid_ns to take both of these pid
> namespaces. The active pid namespace and the default pid namespace
> of children. Allowing me to simply implement unsharing a pid
> namespace in clone after already unsharing a pid namespace with
> unshare.

If clone creates a new namespace, and unshare() discard that namespace
and creates another new one, presumably the first one's reference count
will go to zero if no processes are in it?

Also, when the PID 1 of a namespace leaves that namespace (generally by
exiting), all the children get killed.

I thought the one nasty corner case is that the parent of PID 1 isn't
(ever) in the current PID namespace, so reference counting and list
membership gets a little funky. (I still need to read more about that...)

Rob

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