Subject: Re: [RFC][PATCH] Do not set /proc inode->pid for non-pid-related inodes
Posted by Herbert Poetzl on Mon, 26 Mar 2007 14:36:55 GMT
View Forum Message <> Reply to Message

On Mon, Mar 26, 2007 at 08:54:14AM -0500, Serge E. Hallyn wrote:
> Quoting Herbert Poetzl (herbert@13thfloor.at):
> > On Tue, Mar 20, 2007 at 11:00:57AM -0500, Serge E. Hallyn wrote:
> > > Quoting Eric W. Biederman (ebiederm@xmission.com):
> > > > "Serge E. Hallyn" <serue@us.ibm.com> writes:
> > > >
> > > > > Quoting Eric W. Biederman (ebiederm@xmission.com):
> > > > >> Dave Hansen <hansendc@us.ibm.com> writes:
> > > > >> > On Mon, 2007-03-19 at 20:04 -0600, Eric W. Biederman wrote:
> > > >
> > > > >> >> I would also
> > > > >> >> like to see how we perform the appropriate lookups by pid
> > > > >> >> namespace.
> > > > >> >
> > > > >> > What do you mean?
> > > > >>
> > > > >> proc_pid_readdir ... next_tgid().
> > > > >
> > > > > next_tgid() is simple enough - we can always use current->pid_ns
> > > > > to find the next pidnr.
> > > >
> > > > No. We cannot use current->pid_ns. We must get it from the mount or
> > > > something in the mount.
> > >
> > > Actually I think Dave has it coming from superblock data.
> > >
> > > > Using current to set the default pid_ns to mount is fine. But if
> > > > we use current to select our files we have a moderately serious
> > > > problem.
> > > >
> > > > > The only hitch, as mentioned earlier, is how do we find the first
> > > > > task. Currently task 1 is statically stored as the first inode,
> > > > > and as Dave mentioned we can't do that now, because we dont' know
> > > > > of any one task which will outlive the pid_ns.
> > > >
> > > > Outlive is the wrong concept. Ideally we want something that will
> > > > live as long as there are processes in the pid_ns.
> > >
> > > And there is no such thing.
> > >
> > > > As I thought about this some more there are some problems for
> > > > holding a reference to a pid_ns for a long period of time. Currently
> > > > struct_pid is designed so you can hang onto it forever. struct
> > > > pid_namespace isn't. So we have some very interesting semantic

> > > > questions of what happens when the pid namespace exits.
> > > >
> > > > Since we distinguish mounts by their pid namespace this looks like
> > > > something we need to sort through.
> > >
> > > Yup.
> > >
> > > > >> While I'm not categorically opposed to supporting things like
> > > > >> that it but it is something for which we need to tread very
> > > > >> carefully because it is an extension of current semantics. I
> > > > >> can't think of any weird semantics right now but for something
> > > > >> user visible we will have to support indefinitely I don't see a
> > > > >> reason to rush into it either.
> > > > >
> > > > > Except that unless we mandate that pid1 in any namespace can't
> > > > > exit, and put that feature off until later, we can't not address
> > > > > it.
> > > >
> > > > What if we mandate that pid1 is the last process to exit?
> > >
> > > I think people have complained about that in the past for application
> > > containers, but I really don't see where it hurts anything.
> > >
> > > Cedric, Herbert, did one of you think it would be bad?
> >
> > yes, we (Linux-VServer) consider that bad, because it
> > would not allow to have lightweight containers which
> > do not have a real init process ...
> >
> > e.g. think: 'guest running sshd only'
>
> The way I'm testing pidspaces right now is
>
> ns_exec -c -p /usr/sbin/sshd -p 9999
>
> in which case sshd is pid1. Works fine...
>
> Would it be very limiting to have the first process have to stick
> around? (I'm asking, not criticizing - it's *my* preference that we
> allow pid==1 to exit, but if that's really not advantageous then
> maybe it's not worth fixing the ugly pieces that require it right now
> - afaik right now that's only the fact that PROC_INODE(/proc)->pid
> points to the struct pid for pidnr==1)

again, we basically support 3 different guest models
(regarding init) which probably can be best explained
with an example ...

1) blend through/fake init (from the host system)

```
  USER   PID %CPU %MEM   VSZ  RSS TTY   STAT START   TIME COMMAND
  root     1  6.0  1.9  2036 1096 ?     S    14:24   0:06 init
  root    38  0.7  0.8  2832  448 ?     S    14:26   0:00 sleep 1000
  root    43 50.0  1.2  2536  676 ?     R    14:26   0:00 ps auxwww
```

2) a real init process (running inside the guest with pid=1)

```
  USER   PID %CPU %MEM   VSZ  RSS TTY   STAT START   TIME COMMAND
  root     1  1.6  0.7  2832  444 ?     S    14:26   0:00 sleep 1000
  root    44  0.0  1.2  2536  676 ?     R    14:26   0:00 ps auxwww
```

3) no init process (inside a guest)

```
  USER   PID %CPU %MEM   VSZ  RSS TTY   STAT START   TIME COMMAND
  root    42  0.4  0.7  2828  444 ?     S    14:26   0:00 sleep 1000
  root    45 38.0  1.2  2536  676 ?     R    14:26   0:00 ps auxwww
```

in cases 1) and 3) the 'first' process is in no
way special for the Guest, and must not be treated
special .. it can also go away anytime without
affecting the other guest processes ...

case 2) could in theory handle the pid=1 process
(which might not be the first process, but usually
is a special init process) special, and it would
be acceptable to zap the context when this process
dies off ...

note that the cases 1) and 2) are the most commonly
used cases as most init processes do not handle case
3) yet. still case 3) is important for application
isolation too (which doesn't need any init)

HTC,
Herbert

> -serge