Subject: Re: [PATCH v5] Fix rmmod/read/write races in /proc entries
Posted by Alexey Dobriyan on Mon, 19 Mar 2007 14:56:15 GMT
View Forum Message <> Reply to Message

On Fri, Mar 16, 2007 at 03:50:30AM -0800, Andrew Morton wrote:
> On Fri, 16 Mar 2007 12:16:13 +0300 Alexey Dobriyan <adobriyan@sw.ru> wrote:
> > On Thu, Mar 15, 2007 at 05:53:04PM -0800, Andrew Morton wrote:
> > > My, what a lot of code you have here. I note that nobody can be assed even
> > > reviewing it. Now why is that?
> >
> > I hope, Al could find some time again.
> >
> > > On Sun, 11 Mar 2007 20:04:56 +0300 Alexey Dobriyan <adobriyan@sw.ru> wrote:
> > > > Fix following races:
> > > > ==========
> > > > 1. Write via ->write_proc sleeps in copy_from_user(). Module disappears
> > > > meanwhile. Or, more generically, system call done on /proc file, method
> > > > supplied by module is called, module dissapeares meanwhile.
> > > >
> > > > pde = create_proc_entry()
> > > > if (!pde)
> > > > return -ENOMEM;
> > > > pde->write_proc = ...
> > > > open
> > > > write
> > > > copy_from_user
> > > > pde = create_proc_entry();
> > > > if (!pde) {
> > > > remove_proc_entry();
> > > > return -ENOMEM;
> > > > /* module unloaded */
> > > > }
> > >
> > > We usually fix that race by pinning the module: make whoever registered the
> > > proc entries also register their THIS_MODULE, do a try_module_get() on it
> > > before we start to play with data structures which the module owns.
> > >
> > > Can we do that here?
> >
> > We can, but it will be unreliable:
> >
> > Typical proc entry creation sequence is
> >
> > pde = create_proc_entry(...);
> > if (pde)
> > pde->owner = THIS_MODULE;
> >
> > Right after create_proc_entry() ->owner is NULL, so try_module_get()

> > won't do anything, but proc_delete_inode() could put module which was
> > never getted.
> >
> > This should fixable by always setting ->owner before proc entry is
> > glued to proc entries tree. Something like this:
> >
> > #define create_proc_entry(...) __create_proc_entry(..., THIS_MODULE)
>
> Yes, I was thinking of something like that.
>
> > However, I think it's not enough: delete_module(2) first waits for
> > refcount becoming zero, only then calls modules's exit function which
> > starts removing proc entries. In between, proc entries are accessible
> > and fully-functional, so try_module_get() can again get module and
> > module_put(pde->owner) can happen AFTER module dissapears.
> > What will it put?
> >
> > And how can you fix that? The only way I know is to REMOVE ->owner
> > completely, once we agree on this pde_users/pde_unload_lock stuff.
>
> I think the rmmod code will take care of that.
>
> Once delete_module() has called try_stop_module(), no following
> try_module_get() will succeed. And see that wait_for_zero_refcount() call
> in there which waits for any present users of the module to go away.

See it. So to make all this reliable we need to
a) #define create_proc_entry(...) __create_proc_entry(..., THIS_MODULE)
   and change all users which want to use underscored version.
   Repeat for friends.
b) drag de_get() under proc_subdir_lock
c) drag try_module_get() under proc_subdir_lock

Now there is a problem with ->owner flippers which change ->owner on the
fly, which means try_module_get() and module_put() could be done on
different modules, ick.

One such [easily fixable] user is snd_info_register() which does

```
 p = snd_create_proc_entry(entry->name, entry->mode, root);
 if (!p) {
  mutex_unlock(&info_mutex);
  return -ENOMEM;
 }
 p->owner = entry->module;
```

but snd_create_proc_entry() created proc entry with THIS_MODULE of
sound/core/info.c.

Very little we can do about this except periodically checking every proc entry. With all those pde_users/pde_unload_lock/pde_unload_completion patches I've posted ->owner can go away nicely fixing ->owner flippers problem too. There would be simply nothing to flip.