Subject: Re: [PATCH RESEND 2/2] Fix some kallsyms_lookup() vs rmmod races
Posted by Andrew Morton on Fri, 16 Mar 2007 20:49:49 GMT
View Forum Message <> Reply to Message

On Fri, 16 Mar 2007 20:27:29 +0000 Paulo Marques <pmarques@grupopie.com> wrote:

> Andrew Morton wrote:
> > On Fri, 16 Mar 2007 17:16:39 +0000 Paulo Marques <pmarques@grupopie.com> wrote:
> >
> >> Does freeze_processes() / unfreeze_processes() solve this by only
> >> freezing processes that have voluntarily scheduled (opposed to just
> >> being preempted)?
> >
> > It goes much much further than that. Those processes need to actually
> > perform an explicit call to try_to_freeze().
>
> Ok, I've just done a few tests with the attached patch. It basically
> creates a freeze_machine_run function that is equivalent in interface to
> stop_machine_run, but uses freeze_processes / thaw_processes to stop the
> machine.
>
> This is more of a proof of concept than an actual patch. At the very
> least "freeze_machine_run" should be moved to kernel/power/process.c and
> declared at include/linux/freezer.h so that it could be treated as a
> more general purpose function and not something that is module specific.

OK.

> Anyway, I then tested it by running a modprobe/rmmod loop while running
> a "cat /proc/kallsyms" loop.
>
> On the first run I forgot to remove the mutex_lock(module_mutex) from
> the /proc/kallsyms read path and the freezer was unable to freeze the
> "cat" process that was waiting for the same mutex that the freezer
> process was holding :P
>
> After removing the module_mutex locking from "module_get_kallsym"
> everything was going fine (at least I got no oopses) until I got this:
>
> kernel: Stopping user space processes timed out after 20 seconds (1
> tasks refusing to freeze):
> kernel: kbluetoothd
> kernel: Restarting tasks ... <4> Strange, kseriod not stopped
> kernel: Strange, pdflush not stopped
> kernel: Strange, pdflush not stopped
> kernel: Strange, kswapd0 not stopped
> kernel: Strange, cifsoplockd not stopped
> kernel: Strange, cifsdnotifyd not stopped

> kernel: Strange, jfsIO not stopped
> kernel: Strange, jfsCommit not stopped
> kernel: Strange, jfsCommit not stopped
> kernel: Strange, jfsSync not stopped
> kernel: Strange, xfslogd/0 not stopped
> kernel: Strange, xfslogd/1 not stopped
> kernel: Strange, xfsdatad/0 not stopped
> kernel: Strange, xfsdatad/1 not stopped
> kernel: Strange, kjournald not stopped
> kernel: Strange, khubd not stopped
> kernel: Strange, khelper not stopped
> kernel: Strange, kbluetoothd not stopped
> kernel: done.

There are a bunch of freezer fixes in -mm. But problems might still remain
- I don't think freezer has had a lot of load put on it yet, but it will
soon and it needs to become reliable.

> I repeated the test and did a Alt+SysRq+T to try to find out what
> kbluetoothd was doing and got this:
>
> kernel: kbluetoothd   D 79A11860   0 19156   1   19142
> (NOTLB)
> kernel: 9a269e4c 00000082 00000001 79a11860 00000000 79a09860 c7018030
> 00000003
> kernel: 9a269e71 78475100 c7ebe000 c6730e40 00000000 00000001 00000001
> 00000001
> kernel: 00000000 9a2d7570 79a11860 c7018140 00000000 00001832 42430d03
> 000000ab
> kernel: Call Trace:
> kernel: [<7845dba3>] wait_for_completion+0x7d/0xb7
> kernel: [<781190ba>] default_wake_function+0x0/0xc
> kernel: [<781190ba>] default_wake_function+0x0/0xc
> kernel: [<7812c759>] call_usermodehelper_keys+0xd1/0xf1
> kernel: [<7812c41e>] request_module+0x96/0xd9
> kernel: [<783e30fe>] sock_alloc_inode+0x20/0x4e
> kernel: [<78172559>] alloc_inode+0x15/0x115
> kernel: [<78172d87>] new_inode+0x24/0x81
> kernel: [<783e4003>] __sock_create+0x111/0x199
> kernel: [<783e40a3>] sock_create+0x18/0x1d
> kernel: [<783e40e1>] sys_socket+0x1c/0x43
> kernel: [<783e51da>] sys_socketcall+0x247/0x24c
> kernel: [<78121b2d>] sys_gettimeofday+0x2c/0x65
> kernel: [<78103f10>] sysenter_past_esp+0x5d/0x81
>
> And this was as far as I got...
>

> This actually seems like a better approach than to hold module_mutex
> everywhere to account for an operation that should be "rare" (module
> loading/unloading). If something like this goes in, there are probably a
> few more places inside module.c where we can drop the locking completely.

Yes, using the freezer and module load/unload time seems like a good idea.

> However, it still has a few gotchas. Apart from the problem above (which
> may still be me doing something wrong) it makes module loading /
> unloading depend on CONFIG_PM which is somewhat unexpected for the user.

yup.

> Would it make sense to separate the process freezing / thawing API from
> actual power management and create a new config option (CONFIG_FREEZER?)
> that was automatically selected by the systems that used it (CONFIG_PM,
> CONFIG_MODULES, etc.)? or is that overkill?

Yes, freezer needs to be decoupled from swsusp and from power management
and it should become a first-class core kernel component. Whether we would
need a CONFIG_FREEZER isn't clear - I suspect we'd end up just compiling it
unconditionally.

I cc'ed Rafael, who is doing the freezer revamp work.
