
Subject: Re: [PATCH 1/2] rcfs core patch
Posted by [serue](#) on Sun, 11 Mar 2007 16:36:04 GMT
[View Forum Message](#) <> [Reply to Message](#)

Quoting Herbert Poetzl (herbert@13thfloor.at):

> On Fri, Mar 09, 2007 at 11:27:07PM +0530, Srivatsa Vaddagiri wrote:
> > On Fri, Mar 09, 2007 at 01:38:19AM +0100, Herbert Poetzl wrote:
> > > 2) you allow a task to selectively reshare namespaces/subsystems with
> > > another task, i.e. you can update current->task_proxy to point to
> > > a proxy that matches your existing task_proxy in some ways and the
> > > task_proxy of your destination in others. In that case a trivial
> > > implementation would be to allocate a new task_proxy and copy some
> > > pointers from the old task_proxy and some from the new. But then
> > > whenever a task moves between different groupings it acquires a
> > > new unique task_proxy. So moving a bunch of tasks between two
> > > groupings, they'd all end up with unique task_proxy objects with
> > > identical contents.
>
> > this is exactly what Linux-VServer does right now, and I'm
> > still not convinced that the nsproxy really buys us anything
> > compared to a number of different pointers to various spaces
> > (located in the task struct)
>
> > Are you saying that the current scheme of storing pointers to
> > different spaces (uts_ns, ipc_ns etc) in nsproxy doesn't buy
> > anything?
>
> > Or are you referring to storage of pointers to resource
> > (name)spaces in nsproxy doesn't buy anything?
>
> > In either case, doesn't it buy speed and storage space?
>
> let's do a few examples here, just to illustrate the
> advantages and disadvantages of nsproxy as separate
> structure over nsproxy as part of the task_struct

But you're forgetting the *common* case, which is hundreds or thousands of tasks with just one nsproxy. That's case for which we have to optimize.

When that case is no longer the common case, we can yank the nsproxy. As I keep saying, it *is* just an optimization.

-serge

> 1) typical setup, 100 guests as shell servers, 5
> tasks each when unused, 10 tasks when used 10%
> used in average

- >
- > a) separate nsproxy, we need at least 100
- > structs to handle that (saves some space)
- >
- > we might end up with ~500 nsproxies, if
- > the shell clones a new namespace (so might
- > not save that much space)
- >
- > we do a single inc/dec when the nsproxy
- > is reused, but do the full N inc/dec when
- > we have to copy an nsproxy (might save
- > some refcounting)
- >
- > we need to do the indirection step, from
- > task to nsproxy to space (and data)
- >
- > b) we have ~600 tasks with 600 times the
- > nsproxy data (uses up some more space)
- >
- > we have to do the full N inc/dev when
- > we create a new task (more refcounting)
- >
- > we do not need to do the indirection, we
- > access spaces directly from the 'hot'
- > task struct (makes hot pathes quite fast)
- >
- > so basically we trade a little more space and
- > overhead on task creation for having no
- > indirection to the data accessed quite often
- > throughout the tasks life (hopefully)
- >
- > 2) context migration: for whatever reason, we decide
- > to migrate a task into a subset (space mix) of a
- > context 1000 times
- >
- > a) separate nsproxy, we need to create a new one
- > consisting of the 'new' mix, which will
- >
- > - allocate the nsproxy struct
- > - inc refcounts to all copied spaces
- > - inc refcount nsproxy and assign to task
- > - dec refcount existing task nsproxy
- >
- > after task completion
- > - dec nsproxy refcount
- > - dec refcounts for all spaces
- > - free up nsproxy struct
- >

> b) nsproxy data in task struct
>
> - inc/dec refcounts to changed spaces
>
> after task completion
> - dec refcounts to spaces
>
> so here we gain nothing with the nsproxy, unless
> the chosen subset is identical to the one already
> used, where we end up with a single refcount
> instead of N
>
> > I'd prefer to do accounting (and limits) in a very simple
> > and especially performant way, and the reason for doing
> > so is quite simple:
>
> > Can you elaborate on the relationship between data structures
> > used to store those limits to the task_struct?
>
> sure it is one to many, i.e. each task points to
> exactly one context struct, while a context can
> consist of zero, one or many tasks (no back-
> pointers there)
>
> > Does task_struct store pointers to those objects directly?
>
> it contains a single pointer to the context struct,
> and that contains (as a substruct) the accounting
> and limit information
>
> HTC,
> Herbert
>
> > --
> > Regards,
> > vatsa
> > _____
> > Containers mailing list
> > Containers@lists.osdl.org
> > https://lists.osdl.org/mailman/listinfo/containers
> _____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers
