Subject: Re: [PATCH 1/2] rcfs core patch
Posted by dev on Fri, 09 Mar 2007 09:23:55 GMT
View Forum Message <> Reply to Message

>>There have been various projects attempting to provide resource
>>management support in Linux, including CKRM/Resource Groups and UBC.
>
>
> let me note here, once again, that you forgot Linux-VServer
> which does quite non-intrusive resource management ...
Herbert, do you care to send patches except for ask others to do
something that works for you?

Looks like your main argument is non-intrusive...
"working", "secure", "flexible" are not required to people any more? :/

>> Each had its own task-grouping mechanism.
>
>
> the basic 'context' (pid space) is the grouping mechanism
> we use for resource management too
>
>
>>Paul Menage observed [1] that cpusets in the kernel already has a
>>grouping mechanism which was working well for cpusets. He went ahead
>>and generalized the grouping code in cpusets so that it could be used
>>for overall resource management purpose.
>
>
>>With his patches, it is possible to even create multiple hierarchies
>>of groups (see [2] on why multiple hierarchies) as follows:
>
>
> do we need or even want that? IMHO the hierarchical
> concept CKRM was designed with, was also the reason
> for it being slow, unuseable and complicated
1. cpusets are hierarchical already. So hierarchy is required.
2. As it was discussed on the call controllers which are flat
   can just prohibit creation of hierarchy on the filesystem.
   i.e. allow only 1 depth and continue being fast.

>>mount -t container -o cpuset none /dev/cpuset <- cpuset hierarchy
>>mount -t container -o mem,cpu none /dev/mem <- memory/cpu hierarchy
>>mount -t container -o disk none /dev/disk <- disk hierarchy
>>
>>In each hierarchy, you can create task groups and manipulate the
>>resource parameters of each group. You can also move tasks between
>>groups at run-time (see [3] on why this is required).

>
>
>>Each hierarchy is also manipulated independent of the other.
>
>
>>Paul's patches also introduced a 'struct container' in the kernel,
>>which serves these key purposes:
>>
>>- Task-grouping
>> 'struct container' represents a task-group created in each hierarchy.
>> So every directory created under /dev/cpuset or /dev/mem above will
>> have a corresponding 'struct container' inside the kernel. All tasks
>> pointing to the same 'struct container' are considered to be part of
>> a group
>>
>> The 'struct container' in turn has pointers to resource objects which
>> store actual resource parameters for that group. In above example,
>> 'struct container' created under /dev/cpuset will have a pointer to
>> 'struct cpuset' while 'struct container' created under /dev/disk will
>> have pointer to 'struct disk_quota_or_whatever'.
>>
>>- Maintain hierarchical information
>> The 'struct container' also keeps track of hierarchical relationship
>> between groups.
>>
>>The filesystem interface in the patches essentially serves these
>>purposes:
>>
>> - Provide an interface to manipulate task-groups. This includes
>>   creating/deleting groups, listing tasks present in a group and
>>   moving tasks across groups
>>
>> - Provdes an interface to manipulate the resource objects
>>   (limits etc) pointed to by 'struct container'.
>>
>>As you know, the introduction of 'struct container' was objected
>>to and was felt redundant as a means to group tasks. Thats where I
>>took a shot at converting over Paul Menage's patch to avoid 'struct
>>container' abstraction and insead work with 'struct nsproxy'.
>
>
> which IMHO isn't a step in the right direction, as
> you will need to handle different nsproxies within
> the same 'resource container' (see previous email)
tend to agree.
Looks like Paul's original patch was in the right way.

[...]

>>A separate filesystem would give us more flexibility like the
>>implementing multi-hierarchy support described above.
>
>
> why is the filesystem approach so favored for this
> kind of manipulations?
>
> IMHO it is one of the worst interfaces I can imagine
> (to move tasks between spaces and/or assign resources)
> but yes, I'm aware that filesystems are 'in' nowadays
I also hate filesystems approach being used nowdays everywhere.
But, looks like there are reasons still:
1. cpusets already use fs interface.
2. each controller can have a bit of specific information/controls exported easily.

Can you suggest any other extensible/flexible interface for these?

Thanks,
Kirill

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