
Subject: [PATCH 0/7] Containers (V8): Generic Process Containers

Posted by [Paul Menage](#) on Fri, 06 Apr 2007 23:32:21 GMT

[View Forum Message](#) <> [Reply to Message](#)

--

This is an update to my multi-hierarchy multi-subsystem generic process containers patch. Changes since V7 (12th Feb) include:

- Removed the config-time choice of the number of supported hierarchies - this is now completely dynamic; new hierarchies are allocated on demand, and freed when no longer in use.
- Subsystems are now registered at compile-time in `linux/container_subsys.h`. This allows for faster access to subsystem state since the id is a compile-time constant, so there's only a single extra pointer dereference compared to having a pointer directly in the `task_struct`. It also avoids wasting space with unused subsystem pointers.
- Removed the container pointers from `container_group` - this results in a structure very similar to Srivatsa Vaddagiri's `rcfs` approach. (RCFS uses the `nsproxy` object rather than the `container_group` object; merging `container_group` and `nsproxy` would be pretty straightforward if desired).
- Removed `callback_mutex` from container subsystem to be purely back in the `cpuset` subsystem. Renamed `manage_mutex` to `container_mutex`.
- Condensed `post_attach_task()` into `attach_task()` now that `callback_mutex` is purely within `cpuset.c`
- Simplified the `container_subsys_state` reference counting - stricter rules on liveness make adding reference counts cheaper.

Still TODO:

- decide whether "Containers" is an acceptable name for the system given its usage by some other development groups, or whether something else (ProcessSets? ResourceGroups?) would be better
- decide whether merging `container_group` and `nsproxy` is desirable
- add a hash-table based lookup for `container_group` objects.
- use `seq_file` properly in container tasks files (and also in `cpuset_attach_task`) to avoid having to allocate a big array for all the container's task pointers.

- add back support for the "release agent" functionality
- lots more testing
- define standards for container file names

Generic Process Containers

There have recently been various proposals floating around for resource management/accounting and other task grouping subsystems in the kernel, including ResGroups, User BeanCounters, NSProxy containers, and others. These all need the basic abstraction of being able to group together multiple processes in an aggregate, in order to track/limit the resources permitted to those processes, or control other behaviour of the processes, and all implement this grouping in different ways.

Already existing in the kernel is the cpuset subsystem; this has a process grouping mechanism that is mature, tested, and well documented (particularly with regards to synchronization rules).

This patchset extracts the process grouping code from cpusets into a generic container system, and makes the cpusets code a client of the container system.

It also provides several example clients of the container system, including ResGroups, BeanCounters and namespace proxy.

The change is implemented in three implementation patches, plus four example subsystems that aren't necessarily intended to be merged as part of this patch set, but demonstrate the applicability of the framework.

- 1) extract the process grouping code from cpusets into a standalone system
- 2) remove the process grouping code from cpusets and hook into the container system
- 3) convert the container system to present a generic multi-hierarchy API, and make cpusets a client of that API
- 4) example of a simple CPU accounting container subsystem. Useful as a boilerplate for people implementing their own subsystems.
- 5) example of implementing ResGroups and its numtasks controller over generic containers

6) example of implementing BeanCounters and its numfiles counter over generic containers

7) example of integrating the namespace isolation code (sys_unshare() or various clone flags) with generic containers, allowing virtual servers to take advantage of other resource control efforts.

The intention is that the various resource management and virtualization efforts can also become container clients, with the result that:

- the userspace APIs are (somewhat) normalised
- it's easier to test out e.g. the ResGroups CPU controller in conjunction with the BeanCounters memory controller, or use either of them as the resource-control portion of a virtual server system.
- the additional kernel footprint of any of the competing resource management systems is substantially reduced, since it doesn't need to provide process grouping/containment, hence improving their chances of getting into the kernel

Signed-off-by: Paul Menage <menage@google.com>
