Subject: [PATCH 0/7] containers (V7): Generic Process Containers
Posted by Paul Menage on Mon, 12 Feb 2007 08:15:21 GMT
View Forum Message <> Reply to Message

--

This is an update to my multi-hierarchy multi-subsystem generic
process containers patch. Changes since V6 (22nd December) include:

- updated to 2.6.20

- added more details about multiple hierarchy support in the
  documentation

- reduced the per-task memory overhead to one pointer (previously it
  was one pointer for each hierarchy). Now each task has
  a pointer to a container_group, which holds the pointers to the
  containers (one per active hierarchy) that the task is attached to
  and the associated per-subsystem state (one per active subsystem).
  This container group is shared (with reference counts) between all
  tasks that have the same set of container mappings.

- added API support for binding/unbinding subsystems to/from active
  hierarchies, by remounting with -oremount,<new-subsys-list>. Currently
  this fails with EBUSY if the hierarchy has a child containers; full
  implementation support is left to a later patch.

- added a bind() subsystem callback to indicate when a subsystem is
  moved between hierarchies

- added container_clone(subsys, task), which creates a child container
  for the hierarchy that the specified subsystem is bound to, and
  moves the given task into that container. An example use of this
  would be in sys_unshare, which could, if the namespace container
  subsystem is active, create a child container when the new namespace
  is created.

- temporarily removed the "release agent" support. It's only currently
  used by CPUsets, and intrudes somewhat on the per-container
  reference counting. If necessary it can be re-added, either as a
  generic subsystem feature or a CPUset-specific feature, via a kernel
  thread that periodically polls containers that have been designated
  as notify_on_release to see if they are releasable

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 stages, 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

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>
