Subject: [PATCH 00/29] Rename Containers to Control Groups
Posted by Paul Menage on Tue, 11 Sep 2007 19:52:39 GMT
View Forum Message <> Reply to Message

[Sending to containers@lists.osdl.org for review before sending on to linux-kernel and akpm]

This patchset renames "Task Containers" to "Control Groups", in line with recent discussions.

These patches are drop-in replacements for those of the same names in the current -mm tree:

task-containersv11-basic-task-container-framework.patch
task-containersv11-basic-task-container-framework-fix.patch
task-containersv11-add-tasks-file-interface.patch
task-containersv11-add-fork-exit-hooks.patch
task-containersv11-add-container_clone-interface.patch
task-containersv11-add-procfs-interface.patch
task-containersv11-shared-container-subsystem-group-arrays.patch
task-containersv11-shared-container-subsystem-group-arrays-avoid-lockdep-warning.patch
task-containersv11-shared-container-subsystem-group-arrays-include-fix.patch
task-containersv11-automatic-userspace-notification-of-idle-containers.patch
task-containersv11-make-cpusets-a-client-of-containers.patch
task-containersv11-example-cpu-accounting-subsystem.patch
task-containersv11-simple-task-container-debug-info-subsystem.patch

add-containerstats-v3.patch
add-containerstats-v3-fix.patch

containers-implement-namespace-tracking-subsystem.patch
containers-implement-namespace-tracking-subsystem-fix-order-of-container-subsystems-in-init-kconfig.patch

memory-controller-add-documentation.patch
memory-controller-resource-counters-v7.patch
memory-controller-resource-counters-v7-fix.patch
memory-controller-containers-setup-v7.patch
memory-controller-accounting-setup-v7.patch
memory-controller-memory-accounting-v7.patch
memory-controller-task-migration-v7.patch
memory-controller-add-per-container-lru-and-reclaim-v7.patch
memory-controller-add-per-container-lru-and-reclaim-v7-fix.patch
memory-controller-oom-handling-v7.patch
memory-controller-add-switch-to-control-what-type-of-pages-to-limit-v7.patch
memory-controller-make-page_referenced-container-aware-v7.patch

They are constructed by running the following perl script over the
original versions of the patches in 2.6.23-rc4-mm1:

perl -pi -e 's/subcontainer/child cgroup/g; s/(\b|_)container(\b|_(?!of)|fs|s)/$1cgroup$2/g;
s/CONTAINER(_|S)/CGROUP$1/g; s/Container/Control Group/g; s/css_group/css_set/g;'
patches/*.patch

Owners of other control-group related patches may want to run this
script or some variation of it on their own patches.

--

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 01/29] task containersv11 basic task container framework
Posted by Paul Menage on Tue, 11 Sep 2007 19:52:40 GMT
View Forum Message <> Reply to Message

From: Paul Menage <menage@google.com>

Generic Process Control Groups

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
cgroups, 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.

This patchset provides a framework for tracking and grouping processes
into arbitrary "cgroups" and assigning arbitrary state to those
groupings, in order to control the behaviour of the cgroup as an
aggregate.

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

- the userspace APIs are (somewhat) normalised

- it's easier to test 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

This patch:

Add the main task cgroups framework - the cgroup filesystem, and the
basic structures for tracking membership and associating subsystem state
objects to tasks.

Signed-off-by: Paul Menage <menage@google.com>
Cc: Serge E. Hallyn <serue@us.ibm.com>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Dave Hansen <haveblue@us.ibm.com>
Cc: Balbir Singh <balbir@in.ibm.com>
Cc: Paul Jackson <pj@sgi.com>
Cc: Kirill Korotaev <dev@openvz.org>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: Srivatsa Vaddagiri <vatsa@in.ibm.com>
Cc: Cedric Le Goater <clg@fr.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>

 Documentation/cgroups.txt | 526 ++++++++++++++++
 include/linux/cgroup.h | 214 +++++
 include/linux/cgroup_subsys.h | 10
 include/linux/magic.h | 1
 include/linux/sched.h | 34
 init/Kconfig | 8
 init/main.c | 3
 kernel/Makefile | 1
 kernel/cgroup.c | 1199 ++++++++++++++++++++++++++++++++++++++
 9 files changed, 1995 insertions(+), 1 deletion(-)

diff -puN /dev/null Documentation/cgroups.txt
--- /dev/null
+++ a/Documentation/cgroups.txt
@@ -0,0 +1,526 @@
+   CGROUPS
+   -------
+

+Written by Paul Menage <menage@google.com> based on Documentation/cpusets.txt
+
+Original copyright statements from cpusets.txt:
+Portions Copyright (C) 2004 BULL SA.
+Portions Copyright (c) 2004-2006 Silicon Graphics, Inc.
+Modified by Paul Jackson <pj@sgi.com>
+Modified by Christoph Lameter <clameter@sgi.com>
+
+CONTENTS:
+=========
+
+1. Control Groups
+ 1.1 What are cgroups ?
+ 1.2 Why are cgroups needed ?
+ 1.3 How are cgroups implemented ?
+ 1.4 What does notify_on_release do ?
+ 1.5 How do I use cgroups ?
+2. Usage Examples and Syntax
+ 2.1 Basic Usage
+ 2.2 Attaching processes
+3. Kernel API
+ 3.1 Overview
+ 3.2 Synchronization
+ 3.3 Subsystem API
+4. Questions
+
+1. Control Groups
+==========
+
+1.1 What are cgroups ?
+----------------------
+
+Control Groups provide a mechanism for aggregating/partitioning sets of
+tasks, and all their future children, into hierarchical groups with
+specialized behaviour.
+
+Definitions:
+
+A *cgroup* associates a set of tasks with a set of parameters for one
+or more subsystems.
+
+A *subsystem* is a module that makes use of the task grouping
+facilities provided by cgroups to treat groups of tasks in
+particular ways. A subsystem is typically a "resource controller" that
+schedules a resource or applies per-cgroup limits, but it may be
+anything that wants to act on a group of processes, e.g. a
+virtualization subsystem.
+

+A *hierarchy* is a set of cgroups arranged in a tree, such that
+every task in the system is in exactly one of the cgroups in the
+hierarchy, and a set of subsystems; each subsystem has system-specific
+state attached to each cgroup in the hierarchy. Each hierarchy has
+an instance of the cgroup virtual filesystem associated with it.
+
+At any one time there may be multiple active hierachies of task
+cgroups. Each hierarchy is a partition of all tasks in the system.
+
+User level code may create and destroy cgroups by name in an
+instance of the cgroup virtual file system, specify and query to
+which cgroup a task is assigned, and list the task pids assigned to
+a cgroup. Those creations and assignments only affect the hierarchy
+associated with that instance of the cgroup file system.
+
+On their own, the only use for cgroups is for simple job
+tracking. The intention is that other subsystems hook into the generic
+cgroup support to provide new attributes for cgroups, such as
+accounting/limiting the resources which processes in a cgroup can
+access. For example, cpusets (see Documentation/cpusets.txt) allows
+you to associate a set of CPUs and a set of memory nodes with the
+tasks in each cgroup.
+
+1.2 Why are cgroups needed ?
+----------------------------
+
+There are multiple efforts to provide process aggregations in the
+Linux kernel, mainly for resource tracking purposes. Such efforts
+include cpusets, CKRM/ResGroups, UserBeanCounters, and virtual server
+namespaces. These all require the basic notion of a
+grouping/partitioning of processes, with newly forked processes ending
+in the same group (cgroup) as their parent process.
+
+The kernel cgroup patch provides the minimum essential kernel
+mechanisms required to efficiently implement such groups. It has
+minimal impact on the system fast paths, and provides hooks for
+specific subsystems such as cpusets to provide additional behaviour as
+desired.
+
+Multiple hierarchy support is provided to allow for situations where
+the division of tasks into cgroups is distinctly different for
+different subsystems - having parallel hierarchies allows each
+hierarchy to be a natural division of tasks, without having to handle
+complex combinations of tasks that would be present if several
+unrelated subsystems needed to be forced into the same tree of
+cgroups.
+
+At one extreme, each resource controller or subsystem could be in a

+separate hierarchy; at the other extreme, all subsystems
+would be attached to the same hierarchy.
+
+As an example of a scenario (originally proposed by vatsa@in.ibm.com)
+that can benefit from multiple hierarchies, consider a large
+university server with various users - students, professors, system
+tasks etc. The resource planning for this server could be along the
+following lines:
+
+ CPU : Top cpuset
+ / \
+ CPUSet1 CPUSet2
+ | |
+ (Profs) (Students)
+
+ In addition (system tasks) are attached to topcpuset (so
+ that they can run anywhere) with a limit of 20%
+
+ Memory : Professors (50%), students (30%), system (20%)
+
+ Disk : Prof (50%), students (30%), system (20%)
+
+ Network : WWW browsing (20%), Network File System (60%), others (20%)
+ / \
+ Prof (15%) students (5%)
+
+Browsers like firefox/lynx go into the WWW network class, while (k)nfsd go
+into NFS network class.
+
+At the same time firefox/lynx will share an appropriate CPU/Memory class
+depending on who launched it (prof/student).
+
+With the ability to classify tasks differently for different resources
+(by putting those resource subsystems in different hierarchies) then
+the admin can easily set up a script which receives exec notifications
+and depending on who is launching the browser he can
+
+ # echo browser_pid > /mnt/<restype>/<userclass>/tasks
+
+With only a single hierarchy, he now would potentially have to create
+a separate cgroup for every browser launched and associate it with
+approp network and other resource class. This may lead to
+proliferation of such cgroups.
+
+Also lets say that the administrator would like to give enhanced network
+access temporarily to a student's browser (since it is night and the user
+wants to do online gaming :) OR give one of the students simulation
+apps enhanced CPU power,

+
+With ability to write pids directly to resource classes, its just a
+matter of :
+
+   # echo pid > /mnt/network/<new_class>/tasks
+   (after some time)
+   # echo pid > /mnt/network/<orig_class>/tasks
+
+Without this ability, he would have to split the cgroup into
+multiple separate ones and then associate the new cgroups with the
+new resource classes.
+
+
+
+1.3 How are cgroups implemented ?
+---------------------------------
+
+Control Groups extends the kernel as follows:
+
+ - Each task in the system has a reference-counted pointer to a
+   css_set.
+
+ - A css_set contains a set of reference-counted pointers to
+   cgroup_subsys_state objects, one for each cgroup subsystem
+   registered in the system. There is no direct link from a task to
+   the cgroup of which it's a member in each hierarchy, but this
+   can be determined by following pointers through the
+   cgroup_subsys_state objects. This is because accessing the
+   subsystem state is something that's expected to happen frequently
+   and in performance-critical code, whereas operations that require a
+   task's actual cgroup assignments (in particular, moving between
+   cgroups) are less common.
+
+ - A cgroup hierarchy filesystem can be mounted for browsing and
+   manipulation from user space.
+
+ - You can list all the tasks (by pid) attached to any cgroup.
+
+The implementation of cgroups requires a few, simple hooks
+into the rest of the kernel, none in performance critical paths:
+
+ - in init/main.c, to initialize the root cgroups and initial
+   css_set at system boot.
+
+ - in fork and exit, to attach and detach a task from its css_set.
+
+In addition a new file system, of type "cgroup" may be mounted, to
+enable browsing and modifying the cgroups presently known to the

+kernel. When mounting a cgroup hierarchy, you may specify a
+comma-separated list of subsystems to mount as the filesystem mount
+options. By default, mounting the cgroup filesystem attempts to
+mount a hierarchy containing all registered subsystems.
+
+If an active hierarchy with exactly the same set of subsystems already
+exists, it will be reused for the new mount. If no existing hierarchy
+matches, and any of the requested subsystems are in use in an existing
+hierarchy, the mount will fail with -EBUSY. Otherwise, a new hierarchy
+is activated, associated with the requested subsystems.
+
+It's not currently possible to bind a new subsystem to an active
+cgroup hierarchy, or to unbind a subsystem from an active cgroup
+hierarchy. This may be possible in future, but is fraught with nasty
+error-recovery issues.
+
+When a cgroup filesystem is unmounted, if there are any
+child cgroups created below the top-level cgroup, that hierarchy
+will remain active even though unmounted; if there are no
+child cgroups then the hierarchy will be deactivated.
+
+No new system calls are added for cgroups - all support for
+querying and modifying cgroups is via this cgroup file system.
+
+Each task under /proc has an added file named 'cgroup' displaying,
+for each active hierarchy, the subsystem names and the cgroup name
+as the path relative to the root of the cgroup file system.
+
+Each cgroup is represented by a directory in the cgroup file system
+containing the following files describing that cgroup:
+
+ - tasks: list of tasks (by pid) attached to that cgroup
+ - notify_on_release flag: run /sbin/cgroup_release_agent on exit?
+
+Other subsystems such as cpusets may add additional files in each
+cgroup dir
+
+New cgroups are created using the mkdir system call or shell
+command. The properties of a cgroup, such as its flags, are
+modified by writing to the appropriate file in that cgroups
+directory, as listed above.
+
+The named hierarchical structure of nested cgroups allows partitioning
+a large system into nested, dynamically changeable, "soft-partitions".
+
+The attachment of each task, automatically inherited at fork by any
+children of that task, to a cgroup allows organizing the work load
+on a system into related sets of tasks. A task may be re-attached to

+any other cgroup, if allowed by the permissions on the necessary
+cgroup file system directories.
+
+When a task is moved from one cgroup to another, it gets a new
+css_set pointer - if there's an already existing css_set with the
+desired collection of cgroups then that group is reused, else a new
+css_set is allocated. Note that the current implementation uses a
+linear search to locate an appropriate existing css_set, so isn't
+very efficient. A future version will use a hash table for better
+performance.
+
+The use of a Linux virtual file system (vfs) to represent the
+cgroup hierarchy provides for a familiar permission and name space
+for cgroups, with a minimum of additional kernel code.
+
+1.4 What does notify_on_release do ?
+------------------------------------
+
+*** notify_on_release is disabled in the current patch set. It will be
+*** reactivated in a future patch in a less-intrusive manner
+
+If the notify_on_release flag is enabled (1) in a cgroup, then
+whenever the last task in the cgroup leaves (exits or attaches to
+some other cgroup) and the last child cgroup of that cgroup
+is removed, then the kernel runs the command specified by the contents
+of the "release_agent" file in that hierarchy's root directory,
+supplying the pathname (relative to the mount point of the cgroup
+file system) of the abandoned cgroup. This enables automatic
+removal of abandoned cgroups. The default value of
+notify_on_release in the root cgroup at system boot is disabled
+(0). The default value of other cgroups at creation is the current
+value of their parents notify_on_release setting. The default value of
+a cgroup hierarchy's release_agent path is empty.
+
+1.5 How do I use cgroups ?
+--------------------------
+
+To start a new job that is to be contained within a cgroup, using
+the "cpuset" cgroup subsystem, the steps are something like:
+
+ 1) mkdir /dev/cgroup
+ 2) mount -t cgroup -ocpuset cpuset /dev/cgroup
+ 3) Create the new cgroup by doing mkdir's and write's (or echo's) in
+   the /dev/cgroup virtual file system.
+ 4) Start a task that will be the "founding father" of the new job.
+ 5) Attach that task to the new cgroup by writing its pid to the
+   /dev/cgroup tasks file for that cgroup.
+ 6) fork, exec or clone the job tasks from this founding father task.

+
+For example, the following sequence of commands will setup a cgroup
+named "Charlie", containing just CPUs 2 and 3, and Memory Node 1,
+and then start a subshell 'sh' in that cgroup:
+
+ mount -t cgroup cpuset -ocpuset /dev/cgroup
+ cd /dev/cgroup
+ mkdir Charlie
+ cd Charlie
+ /bin/echo 2-3 > cpus
+ /bin/echo 1 > mems
+ /bin/echo $$ > tasks
+ sh
+ # The subshell 'sh' is now running in cgroup Charlie
+ # The next line should display '/Charlie'
+ cat /proc/self/cgroup
+
+2. Usage Examples and Syntax
+============================
+
+2.1 Basic Usage
+---------------
+
+Creating, modifying, using the cgroups can be done through the cgroup
+virtual filesystem.
+
+To mount a cgroup hierarchy will all available subsystems, type:
+# mount -t cgroup xxx /dev/cgroup
+
+The "xxx" is not interpreted by the cgroup code, but will appear in
+/proc/mounts so may be any useful identifying string that you like.
+
+To mount a cgroup hierarchy with just the cpuset and numtasks
+subsystems, type:
+# mount -t cgroup -o cpuset,numtasks hier1 /dev/cgroup
+
+To change the set of subsystems bound to a mounted hierarchy, just
+remount with different options:
+
+# mount -o remount,cpuset,ns /dev/cgroup
+
+Note that changing the set of subsystems is currently only supported
+when the hierarchy consists of a single (root) cgroup. Supporting
+the ability to arbitrarily bind/unbind subsystems from an existing
+cgroup hierarchy is intended to be implemented in the future.
+
+Then under /dev/cgroup you can find a tree that corresponds to the
+tree of the cgroups in the system. For instance, /dev/cgroup

+is the cgroup that holds the whole system.
+
+If you want to create a new cgroup under /dev/cgroup:
+# cd /dev/cgroup
+# mkdir my_cgroup
+
+Now you want to do something with this cgroup.
+# cd my_cgroup
+
+In this directory you can find several files:
+# ls
+notify_on_release release_agent tasks
+(plus whatever files are added by the attached subsystems)
+
+Now attach your shell to this cgroup:
+# /bin/echo $$ > tasks
+
+You can also create cgroups inside your cgroup by using mkdir in this
+directory.
+# mkdir my_sub_cs
+
+To remove a cgroup, just use rmdir:
+# rmdir my_sub_cs
+
+This will fail if the cgroup is in use (has cgroups inside, or
+has processes attached, or is held alive by other subsystem-specific
+reference).
+
+2.2 Attaching processes
+----------------------
+
+# /bin/echo PID > tasks
+
+Note that it is PID, not PIDs. You can only attach ONE task at a time.
+If you have several tasks to attach, you have to do it one after another:
+
+# /bin/echo PID1 > tasks
+# /bin/echo PID2 > tasks
+ ...
+# /bin/echo PIDn > tasks
+
+3. Kernel API
+=============
+
+3.1 Overview
+------------
+
+Each kernel subsystem that wants to hook into the generic cgroup

+system needs to create a cgroup_subsys object. This contains
+various methods, which are callbacks from the cgroup system, along
+with a subsystem id which will be assigned by the cgroup system.
+
+Other fields in the cgroup_subsys object include:
+
+- subsys_id: a unique array index for the subsystem, indicating which
+ entry in cgroup->subsys[] this subsystem should be
+ managing. Initialized by cgroup_register_subsys(); prior to this
+ it should be initialized to -1
+
+- hierarchy: an index indicating which hierarchy, if any, this
+ subsystem is currently attached to. If this is -1, then the
+ subsystem is not attached to any hierarchy, and all tasks should be
+ considered to be members of the subsystem's top_cgroup. It should
+ be initialized to -1.
+
+- name: should be initialized to a unique subsystem name prior to
+ calling cgroup_register_subsystem. Should be no longer than
+ MAX_CGROUP_TYPE_NAMELEN
+
+Each cgroup object created by the system has an array of pointers,
+indexed by subsystem id; this pointer is entirely managed by the
+subsystem; the generic cgroup code will never touch this pointer.
+
+3.2 Synchronization
+-------------------
+
+There is a global mutex, cgroup_mutex, used by the cgroup
+system. This should be taken by anything that wants to modify a
+cgroup. It may also be taken to prevent cgroups from being
+modified, but more specific locks may be more appropriate in that
+situation.
+
+See kernel/cgroup.c for more details.
+
+Subsystems can take/release the cgroup_mutex via the functions
+cgroup_lock()/cgroup_unlock(), and can
+take/release the callback_mutex via the functions
+cgroup_lock()/cgroup_unlock().
+
+Accessing a task's cgroup pointer may be done in the following ways:
+- while holding cgroup_mutex
+- while holding the task's alloc_lock (via task_lock())
+- inside an rcu_read_lock() section via rcu_dereference()
+
+3.3 Subsystem API
+-------------------------

+
+Each subsystem should:
+
+- add an entry in linux/cgroup_subsys.h
+- define a cgroup_subsys object called <name>_subsys
+
+Each subsystem may export the following methods. The only mandatory
+methods are create/destroy. Any others that are null are presumed to
+be successful no-ops.
+
+struct cgroup_subsys_state *create(struct cgroup *cont)
+LL=cgroup_mutex
+
+Called to create a subsystem state object for a cgroup. The
+subsystem should allocate its subsystem state object for the passed
+cgroup, returning a pointer to the new object on success or a
+negative error code. On success, the subsystem pointer should point to
+a structure of type cgroup_subsys_state (typically embedded in a
+larger subsystem-specific object), which will be initialized by the
+cgroup system. Note that this will be called at initialization to
+create the root subsystem state for this subsystem; this case can be
+identified by the passed cgroup object having a NULL parent (since
+it's the root of the hierarchy) and may be an appropriate place for
+initialization code.
+
+void destroy(struct cgroup *cont)
+LL=cgroup_mutex
+
+The cgroup system is about to destroy the passed cgroup; the
+subsystem should do any necessary cleanup
+
+int can_attach(struct cgroup_subsys *ss, struct cgroup *cont,
+	 struct task_struct *task)
+LL=cgroup_mutex
+
+Called prior to moving a task into a cgroup; if the subsystem
+returns an error, this will abort the attach operation. If a NULL
+task is passed, then a successful result indicates that *any*
+unspecified task can be moved into the cgroup. Note that this isn't
+called on a fork. If this method returns 0 (success) then this should
+remain valid while the caller holds cgroup_mutex.
+
+void attach(struct cgroup_subsys *ss, struct cgroup *cont,
+	 struct cgroup *old_cont, struct task_struct *task)
+LL=cgroup_mutex
+
+
+Called after the task has been attached to the cgroup, to allow any

+post-attachment activity that requires memory allocations or blocking.
+
+void fork(struct cgroup_subsy *ss, struct task_struct *task)
+LL=callback_mutex, maybe read_lock(tasklist_lock)
+
+Called when a task is forked into a cgroup. Also called during
+registration for all existing tasks.
+
+void exit(struct cgroup_subsys *ss, struct task_struct *task)
+LL=callback_mutex
+
+Called during task exit
+
+int populate(struct cgroup_subsys *ss, struct cgroup *cont)
+LL=none
+
+Called after creation of a cgroup to allow a subsystem to populate
+the cgroup directory with file entries. The subsystem should make
+calls to cgroup_add_file() with objects of type cftype (see
+include/linux/cgroup.h for details). Note that although this
+method can return an error code, the error code is currently not
+always handled well.
+
+void bind(struct cgroup_subsys *ss, struct cgroup *root)
+LL=callback_mutex
+
+Called when a cgroup subsystem is rebound to a different hierarchy
+and root cgroup. Currently this will only involve movement between
+the default hierarchy (which never has sub-cgroups) and a hierarchy
+that is being created/destroyed (and hence has no sub-cgroups).
+
+4. Questions
+============
+
+Q: what's up with this '/bin/echo' ?
+A: bash's builtin 'echo' command does not check calls to write() against
+   errors. If you use it in the cgroup file system, you won't be
+   able to tell whether a command succeeded or failed.
+
+Q: When I attach processes, only the first of the line gets really attached !
+A: We can only return one error code per call to write(). So you should also
+   put only ONE pid.
+
diff -puN /dev/null include/linux/cgroup.h
--- /dev/null
+++ a/include/linux/cgroup.h
@@ -0,0 +1,214 @@
+#ifndef _LINUX_CGROUP_H

```
+#define _LINUX_CGROUP_H
+/*
+ *  cgroup interface
+ *
+ *  Copyright (C) 2003 BULL SA
+ *  Copyright (C) 2004-2006 Silicon Graphics, Inc.
+ *
+ */
+
+#include <linux/sched.h>
+#include <linux/kref.h>
+#include <linux/cpumask.h>
+#include <linux/nodemask.h>
+#include <linux/rcupdate.h>
+
+#ifdef CONFIG_CGROUPS
+
+struct cgroupfs_root;
+struct cgroup_subsys;
+struct inode;
+
+extern int cgroup_init_early(void);
+extern int cgroup_init(void);
+extern void cgroup_init_smp(void);
+extern void cgroup_lock(void);
+extern void cgroup_unlock(void);
+
+/* Per-subsystem/per-cgroup state maintained by the system. */
+struct cgroup_subsys_state {
+ /* The cgroup that this subsystem is attached to. Useful
+  * for subsystems that want to know about the cgroup
+  * hierarchy structure */
+ struct cgroup *cgroup;
+
+ /* State maintained by the cgroup system to allow
+  * subsystems to be "busy". Should be accessed via css_get()
+  * and css_put() */
+
+ atomic_t refcnt;
+
+ unsigned long flags;
+};
+
+/* bits in struct cgroup_subsys_state flags field */
+enum {
+ CSS_ROOT, /* This CSS is the root of the subsystem */
+};
+
```

```
+/*
+ * Call css_get() to hold a reference on the cgroup;
+ *
+ */
+
+static inline void css_get(struct cgroup_subsys_state *css)
+{
+ /* We don't need to reference count the root state */
+ if (!test_bit(CSS_ROOT, &css->flags))
+  atomic_inc(&css->refcnt);
+}
+/*
+ * css_put() should be called to release a reference taken by
+ * css_get()
+ */
+
+static inline void css_put(struct cgroup_subsys_state *css)
+{
+ if (!test_bit(CSS_ROOT, &css->flags))
+  atomic_dec(&css->refcnt);
+}
+
+struct cgroup {
+ unsigned long flags;  /* "unsigned long" so bitops work */
+
+ /* count users of this cgroup. >0 means busy, but doesn't
+  * necessarily indicate the number of tasks in the
+  * cgroup */
+ atomic_t count;
+
+ /*
+  * We link our 'sibling' struct into our parent's 'children'.
+  * Our children link their 'sibling' into our 'children'.
+  */
+ struct list_head sibling; /* my parent's children */
+ struct list_head children; /* my children */
+
+ struct cgroup *parent; /* my parent */
+ struct dentry *dentry;    /* cgroup fs entry */
+
+ /* Private pointers for each registered subsystem */
+ struct cgroup_subsys_state *subsys[CGROUP_SUBSYS_COUNT];
+
+ struct cgroupfs_root *root;
+ struct cgroup *top_cgroup;
+};
+
+/* struct cftype:
```

```
+ *
+ * The files in the cgroup filesystem mostly have a very simple read/write
+ * handling, some common function will take care of it. Nevertheless some cases
+ * (read tasks) are special and therefore I define this structure for every
+ * kind of file.
+ *
+ *
+ * When reading/writing to a file:
+ * - the cgroup to use in file->f_dentry->d_parent->d_fsdata
+ * - the 'cftype' of the file is file->f_dentry->d_fsdata
+ */
+
+#define MAX_CFTYPE_NAME 64
+struct cftype {
+ /* By convention, the name should begin with the name of the
+  * subsystem, followed by a period */
+ char name[MAX_CFTYPE_NAME];
+ int private;
+ int (*open) (struct inode *inode, struct file *file);
+ ssize_t (*read) (struct cgroup *cont, struct cftype *cft,
+    struct file *file,
+    char __user *buf, size_t nbytes, loff_t *ppos);
+ /*
+  * read_uint() is a shortcut for the common case of returning a
+  * single integer. Use it in place of read()
+  */
+ u64 (*read_uint) (struct cgroup *cont, struct cftype *cft);
+ ssize_t (*write) (struct cgroup *cont, struct cftype *cft,
+     struct file *file,
+     const char __user *buf, size_t nbytes, loff_t *ppos);
+ int (*release) (struct inode *inode, struct file *file);
+};
+
+/* Add a new file to the given cgroup directory. Should only be
+ * called by subsystems from within a populate() method */
+int cgroup_add_file(struct cgroup *cont, struct cgroup_subsys *subsys,
+         const struct cftype *cft);
+
+/* Add a set of new files to the given cgroup directory. Should
+ * only be called by subsystems from within a populate() method */
+int cgroup_add_files(struct cgroup *cont,
+   struct cgroup_subsys *subsys,
+   const struct cftype cft[],
+   int count);
+
+int cgroup_is_removed(const struct cgroup *cont);
+
+int cgroup_path(const struct cgroup *cont, char *buf, int buflen);
```

```
+
+/* Return true if the cgroup is a descendant of the current cgroup */
+int cgroup_is_descendant(const struct cgroup *cont);
+
+/* Control Group subsystem type. See Documentation/cgroups.txt for details */
+
+struct cgroup_subsys {
+ struct cgroup_subsys_state *(*create)(struct cgroup_subsys *ss,
+        struct cgroup *cont);
+ void (*destroy)(struct cgroup_subsys *ss, struct cgroup *cont);
+ int (*can_attach)(struct cgroup_subsys *ss,
+     struct cgroup *cont, struct task_struct *tsk);
+ void (*attach)(struct cgroup_subsys *ss, struct cgroup *cont,
+   struct cgroup *old_cont, struct task_struct *tsk);
+ void (*fork)(struct cgroup_subsys *ss, struct task_struct *task);
+ void (*exit)(struct cgroup_subsys *ss, struct task_struct *task);
+ int (*populate)(struct cgroup_subsys *ss,
+   struct cgroup *cont);
+ void (*bind)(struct cgroup_subsys *ss, struct cgroup *root);
+ int subsys_id;
+ int active;
+ int early_init;
+#define MAX_CGROUP_TYPE_NAMELEN 32
+ const char *name;
+
+ /* Protected by RCU */
+ struct cgroupfs_root *root;
+
+ struct list_head sibling;
+
+ void *private;
+};
+
+#define SUBSYS(_x) extern struct cgroup_subsys _x ## _subsys;
+#include <linux/cgroup_subsys.h>
+#undef SUBSYS
+
+static inline struct cgroup_subsys_state *cgroup_subsys_state(
+ struct cgroup *cont, int subsys_id)
+{
+ return cont->subsys[subsys_id];
+}
+
+static inline struct cgroup_subsys_state *task_subsys_state(
+ struct task_struct *task, int subsys_id)
+{
+ return rcu_dereference(task->cgroups.subsys[subsys_id]);
+}
```

```
+
+static inline struct cgroup* task_cgroup(struct task_struct *task,
+            int subsys_id)
+{
+ return task_subsys_state(task, subsys_id)->cgroup;
+}
+
+int cgroup_path(const struct cgroup *cont, char *buf, int buflen);
+
+#else /* !CONFIG_CGROUPS */
+
+static inline int cgroup_init_early(void) { return 0; }
+static inline int cgroup_init(void) { return 0; }
+static inline void cgroup_init_smp(void) {}
+
+static inline void cgroup_lock(void) {}
+static inline void cgroup_unlock(void) {}
+
+#endif /* !CONFIG_CGROUPS */
+
+#endif /* _LINUX_CGROUP_H */
diff -puN /dev/null include/linux/cgroup_subsys.h
--- /dev/null
+++ a/include/linux/cgroup_subsys.h
@@ -0,0 +1,10 @@
+/* Add subsystem definitions of the form SUBSYS(<name>) in this
+ * file. Surround each one by a line of comment markers so that
+ * patches don't collide
+ */
+
+/* */
+
+/* */
+
+/* */
diff -puN include/linux/magic.h~task-cgroupsv11-basic-task-cgroup-framework
include/linux/magic.h
--- a/include/linux/magic.h~task-cgroupsv11-basic-task-cgroup-framework
+++ a/include/linux/magic.h
@@ -40,5 +40,6 @@

 #define SMB_SUPER_MAGIC  0x517B
 #define USBDEVICE_SUPER_MAGIC 0x9fa2
+#define CGROUP_SUPER_MAGIC 0x27e0eb

 #endif /* __LINUX_MAGIC_H__ */
diff -puN include/linux/sched.h~task-cgroupsv11-basic-task-cgroup-framework
include/linux/sched.h
```

```
--- a/include/linux/sched.h~task-cgroupsv11-basic-task-cgroup-framework
+++ a/include/linux/sched.h
@@ -861,6 +861,34 @@ struct sched_entity {
 #endif
 };

+#ifdef CONFIG_CGROUPS
+
+#define SUBSYS(_x) _x ## _subsys_id,
+enum cgroup_subsys_id {
+#include <linux/cgroup_subsys.h>
+ CGROUP_SUBSYS_COUNT
+};
+#undef SUBSYS
+
+/* A css_set is a structure holding pointers to a set of
+ * cgroup_subsys_state objects.
+ */
+
+struct css_set {
+
+ /* Set of subsystem states, one for each subsystem. NULL for
+  * subsystems that aren't part of this hierarchy. These
+  * pointers reduce the number of dereferences required to get
+  * from a task to its state for a given cgroup, but result
+  * in increased space usage if tasks are in wildly different
+  * groupings across different hierarchies. This array is
+  * immutable after creation */
+ struct cgroup_subsys_state *subsys[CGROUP_SUBSYS_COUNT];
+
+};
+
+#endif /* CONFIG_CGROUPS */
+
 struct task_struct {
  volatile long state; /* -1 unrunnable, 0 runnable, >0 stopped */
  void *stack;
@@ -1096,6 +1124,9 @@ struct task_struct {
  int cpuset_mems_generation;
  int cpuset_mem_spread_rotor;
 #endif
+#ifdef CONFIG_CGROUPS
+ struct css_set cgroups;
+#endif
 #ifdef CONFIG_FUTEX
  struct robust_list_head __user *robust_list;
 #ifdef CONFIG_COMPAT
@@ -1585,7 +1616,8 @@ static inline int thread_group_empty(str
```

```
 /*
  * Protects ->fs, ->files, ->mm, ->group_info, ->comm, keyring
  * subscriptions and synchronises with wait4().  Also used in procfs.  Also
- * pins the final release of task.io_context.  Also protects ->cpuset.
+ * pins the final release of task.io_context.  Also protects ->cpuset and
+ * ->cgroup.subsys[].
  *
  * Nests both inside and outside of read_lock(&tasklist_lock).
  * It must not be nested with write_lock_irq(&tasklist_lock),
diff -puN init/Kconfig~task-cgroupsv11-basic-task-cgroup-framework init/Kconfig
--- a/init/Kconfig~task-cgroupsv11-basic-task-cgroup-framework
+++ a/init/Kconfig
@@ -274,6 +274,14 @@ config LOG_BUF_SHIFT
        13 =>  8 KB
        12 =>  4 KB

+config CGROUPS
+ bool "Control Group support"
+ help
+   This option will let you use process cgroup subsystems
+   such as Cpusets
+
+   Say N if unsure.
+
 config CPUSETS
  bool "Cpuset support"
  depends on SMP
diff -puN init/main.c~task-cgroupsv11-basic-task-cgroup-framework init/main.c
--- a/init/main.c~task-cgroupsv11-basic-task-cgroup-framework
+++ a/init/main.c
@@ -39,6 +39,7 @@
 #include <linux/writeback.h>
 #include <linux/cpu.h>
 #include <linux/cpuset.h>
+#include <linux/cgroup.h>
 #include <linux/efi.h>
 #include <linux/tick.h>
 #include <linux/interrupt.h>
@@ -523,6 +524,7 @@ asmlinkage void __init start_kernel(void
  */
  unwind_init();
  lockdep_init();
+ cgroup_init_early();

  local_irq_disable();
  early_boot_irqs_off();
@@ -640,6 +642,7 @@ asmlinkage void __init start_kernel(void
 #ifdef CONFIG_PROC_FS
```

```
  proc_root_init();
 #endif
+ cgroup_init();
  cpuset_init();
  taskstats_init_early();
  delayacct_init();
diff -puN kernel/Makefile~task-cgroupsv11-basic-task-cgroup-framework kernel/Makefile
--- a/kernel/Makefile~task-cgroupsv11-basic-task-cgroup-framework
+++ a/kernel/Makefile
@@ -38,6 +38,7 @@ obj-$(CONFIG_PM) += power/
 obj-$(CONFIG_BSD_PROCESS_ACCT) += acct.o
 obj-$(CONFIG_KEXEC) += kexec.o
 obj-$(CONFIG_COMPAT) += compat.o
+obj-$(CONFIG_CGROUPS) += cgroup.o
 obj-$(CONFIG_CPUSETS) += cpuset.o
 obj-$(CONFIG_IKCONFIG) += configs.o
 obj-$(CONFIG_STOP_MACHINE) += stop_machine.o
diff -puN /dev/null kernel/cgroup.c
--- /dev/null
+++ a/kernel/cgroup.c
@@ -0,0 +1,1199 @@
+/*
+ *  kernel/cgroup.c
+ *
+ *  Generic process-grouping system.
+ *
+ *  Based originally on the cpuset system, extracted by Paul Menage
+ *  Copyright (C) 2006 Google, Inc
+ *
+ *  Copyright notices from the original cpuset code:
+ *  --------------------------------------------------
+ *  Copyright (C) 2003 BULL SA.
+ *  Copyright (C) 2004-2006 Silicon Graphics, Inc.
+ *
+ *  Portions derived from Patrick Mochel's sysfs code.
+ *  sysfs is Copyright (c) 2001-3 Patrick Mochel
+ *
+ *  2003-10-10 Written by Simon Derr.
+ *  2003-10-22 Updates by Stephen Hemminger.
+ *  2004 May-July Rework by Paul Jackson.
+ *  ---------------------------------------------------
+ *
+ *  This file is subject to the terms and conditions of the GNU General Public
+ *  License.  See the file COPYING in the main directory of the Linux
+ *  distribution for more details.
+ */
+
+#include <linux/cgroup.h>
```

```
+#include <linux/errno.h>
+#include <linux/fs.h>
+#include <linux/kernel.h>
+#include <linux/list.h>
+#include <linux/mm.h>
+#include <linux/mutex.h>
+#include <linux/mount.h>
+#include <linux/pagemap.h>
+#include <linux/rcupdate.h>
+#include <linux/sched.h>
+#include <linux/seq_file.h>
+#include <linux/slab.h>
+#include <linux/magic.h>
+#include <linux/spinlock.h>
+#include <linux/string.h>
+
+#include <asm/atomic.h>
+
+/* Generate an array of cgroup subsystem pointers */
+#define SUBSYS(_x) &_x ## _subsys,
+
+static struct cgroup_subsys *subsys[] = {
+#include <linux/cgroup_subsys.h>
+};
+
+/*
+ * A cgroupfs_root represents the root of a cgroup hierarchy,
+ * and may be associated with a superblock to form an active
+ * hierarchy
+ */
+struct cgroupfs_root {
+ struct super_block *sb;
+
+ /*
+  * The bitmask of subsystems intended to be attached to this
+  * hierarchy
+  */
+ unsigned long subsys_bits;
+
+ /* The bitmask of subsystems currently attached to this hierarchy */
+ unsigned long actual_subsys_bits;
+
+ /* A list running through the attached subsystems */
+ struct list_head subsys_list;
+
+ /* The root cgroup for this hierarchy */
+ struct cgroup top_cgroup;
+
```

```
+ /* Tracks how many cgroups are currently defined in hierarchy.*/
+ int number_of_cgroups;
+
+ /* A list running through the mounted hierarchies */
+ struct list_head root_list;
+
+ /* Hierarchy-specific flags */
+ unsigned long flags;
+};
+
+
+/*
+ * The "rootnode" hierarchy is the "dummy hierarchy", reserved for the
+ * subsystems that are otherwise unattached - it never has more than a
+ * single cgroup, and all tasks are part of that cgroup.
+ */
+static struct cgroupfs_root rootnode;
+
+/* The list of hierarchy roots */
+
+static LIST_HEAD(roots);
+
+/* dummytop is a shorthand for the dummy hierarchy's top cgroup */
+#define dummytop (&rootnode.top_cgroup)
+
+/* This flag indicates whether tasks in the fork and exit paths should
+ * take callback_mutex and check for fork/exit handlers to call. This
+ * avoids us having to do extra work in the fork/exit path if none of the
+ * subsystems need to be called.
+ */
+static int need_forkexit_callback;
+
+/* bits in struct cgroup flags field */
+enum {
+ CONT_REMOVED,
+};
+
+/* convenient tests for these bits */
+inline int cgroup_is_removed(const struct cgroup *cont)
+{
+ return test_bit(CONT_REMOVED, &cont->flags);
+}
+
+/* bits in struct cgroupfs_root flags field */
+enum {
+ ROOT_NOPREFIX, /* mounted subsystems have no named prefix */
+};
+
```

```
+/*
+ * for_each_subsys() allows you to iterate on each subsystem attached to
+ * an active hierarchy
+ */
+#define for_each_subsys(_root, _ss) \
+list_for_each_entry(_ss, &_root->subsys_list, sibling)
+
+/* for_each_root() allows you to iterate across the active hierarchies */
+#define for_each_root(_root) \
+list_for_each_entry(_root, &roots, root_list)
+
+/*
+ * There is one global cgroup mutex. We also require taking
+ * task_lock() when dereferencing a task's cgroup subsys pointers.
+ * See "The task_lock() exception", at the end of this comment.
+ *
+ * A task must hold cgroup_mutex to modify cgroups.
+ *
+ * Any task can increment and decrement the count field without lock.
+ * So in general, code holding cgroup_mutex can't rely on the count
+ * field not changing.  However, if the count goes to zero, then only
+ * attach_task() can increment it again.  Because a count of zero
+ * means that no tasks are currently attached, therefore there is no
+ * way a task attached to that cgroup can fork (the other way to
+ * increment the count).  So code holding cgroup_mutex can safely
+ * assume that if the count is zero, it will stay zero. Similarly, if
+ * a task holds cgroup_mutex on a cgroup with zero count, it
+ * knows that the cgroup won't be removed, as cgroup_rmdir()
+ * needs that mutex.
+ *
+ * The cgroup_common_file_write handler for operations that modify
+ * the cgroup hierarchy holds cgroup_mutex across the entire operation,
+ * single threading all such cgroup modifications across the system.
+ *
+ * The fork and exit callbacks cgroup_fork() and cgroup_exit(), don't
+ * (usually) take cgroup_mutex.  These are the two most performance
+ * critical pieces of code here.  The exception occurs on cgroup_exit(),
+ * when a task in a notify_on_release cgroup exits.  Then cgroup_mutex
+ * is taken, and if the cgroup count is zero, a usermode call made
+ * to /sbin/cgroup_release_agent with the name of the cgroup (path
+ * relative to the root of cgroup file system) as the argument.
+ *
+ * A cgroup can only be deleted if both its 'count' of using tasks
+ * is zero, and its list of 'children' cgroups is empty.  Since all
+ * tasks in the system use _some_ cgroup, and since there is always at
+ * least one task in the system (init, pid == 1), therefore, top_cgroup
+ * always has either children cgroups and/or using tasks.  So we don't
+ * need a special hack to ensure that top_cgroup cannot be deleted.
```

```
+ *
+ * The task_lock() exception
+ *
+ * The need for this exception arises from the action of
+ * attach_task(), which overwrites one tasks cgroup pointer with
+ * another.  It does so using cgroup_mutexe, however there are
+ * several performance critical places that need to reference
+ * task->cgroup without the expense of grabbing a system global
+ * mutex.  Therefore except as noted below, when dereferencing or, as
+ * in attach_task(), modifying a task'ss cgroup pointer we use
+ * task_lock(), which acts on a spinlock (task->alloc_lock) already in
+ * the task_struct routinely used for such matters.
+ *
+ * P.S.  One more locking exception.  RCU is used to guard the
+ * update of a tasks cgroup pointer by attach_task()
+ */
+
+static DEFINE_MUTEX(cgroup_mutex);
+
+/**
+ * cgroup_lock - lock out any changes to cgroup structures
+ *
+ */
+
+void cgroup_lock(void)
+{
+ mutex_lock(&cgroup_mutex);
+}
+
+/**
+ * cgroup_unlock - release lock on cgroup changes
+ *
+ * Undo the lock taken in a previous cgroup_lock() call.
+ */
+
+void cgroup_unlock(void)
+{
+ mutex_unlock(&cgroup_mutex);
+}
+
+/*
+ * A couple of forward declarations required, due to cyclic reference loop:
+ * cgroup_mkdir -> cgroup_create -> cgroup_populate_dir ->
+ * cgroup_add_file -> cgroup_create_file -> cgroup_dir_inode_operations
+ * -> cgroup_mkdir.
+ */
+
+static int cgroup_mkdir(struct inode *dir, struct dentry *dentry, int mode);
```

```
+static int cgroup_rmdir(struct inode *unused_dir, struct dentry *dentry);
+static int cgroup_populate_dir(struct cgroup *cont);
+static struct inode_operations cgroup_dir_inode_operations;
+
+static struct inode *cgroup_new_inode(mode_t mode, struct super_block *sb)
+{
+ struct inode *inode = new_inode(sb);
+ static struct backing_dev_info cgroup_backing_dev_info = {
+  .capabilities = BDI_CAP_NO_ACCT_DIRTY | BDI_CAP_NO_WRITEBACK,
+ };
+
+ if (inode) {
+  inode->i_mode = mode;
+  inode->i_uid = current->fsuid;
+  inode->i_gid = current->fsgid;
+  inode->i_blocks = 0;
+  inode->i_atime = inode->i_mtime = inode->i_ctime = CURRENT_TIME;
+  inode->i_mapping->backing_dev_info = &cgroup_backing_dev_info;
+ }
+ return inode;
+}
+
+static void cgroup_diput(struct dentry *dentry, struct inode *inode)
+{
+ /* is dentry a directory ? if so, kfree() associated cgroup */
+ if (S_ISDIR(inode->i_mode)) {
+  struct cgroup *cont = dentry->d_fsdata;
+  BUG_ON(!(cgroup_is_removed(cont)));
+  kfree(cont);
+ }
+ iput(inode);
+}
+
+static struct dentry *cgroup_get_dentry(struct dentry *parent,
+        const char *name)
+{
+ struct dentry *d = lookup_one_len(name, parent, strlen(name));
+ static struct dentry_operations cgroup_dops = {
+  .d_iput = cgroup_diput,
+ };
+
+ if (!IS_ERR(d))
+  d->d_op = &cgroup_dops;
+ return d;
+}
+
+static void remove_dir(struct dentry *d)
+{
```

```
+ struct dentry *parent = dget(d->d_parent);
+
+ d_delete(d);
+ simple_rmdir(parent->d_inode, d);
+ dput(parent);
+}
+
+static void cgroup_clear_directory(struct dentry *dentry)
+{
+ struct list_head *node;
+
+ BUG_ON(!mutex_is_locked(&dentry->d_inode->i_mutex));
+ spin_lock(&dcache_lock);
+ node = dentry->d_subdirs.next;
+ while (node != &dentry->d_subdirs) {
+  struct dentry *d = list_entry(node, struct dentry, d_u.d_child);
+  list_del_init(node);
+  if (d->d_inode) {
+   /* This should never be called on a cgroup
+    * directory with child cgroups */
+   BUG_ON(d->d_inode->i_mode & S_IFDIR);
+   d = dget_locked(d);
+   spin_unlock(&dcache_lock);
+   d_delete(d);
+   simple_unlink(dentry->d_inode, d);
+   dput(d);
+   spin_lock(&dcache_lock);
+  }
+  node = dentry->d_subdirs.next;
+ }
+ spin_unlock(&dcache_lock);
+}
+
+/*
+ * NOTE : the dentry must have been dget()'ed
+ */
+static void cgroup_d_remove_dir(struct dentry *dentry)
+{
+ cgroup_clear_directory(dentry);
+
+ spin_lock(&dcache_lock);
+ list_del_init(&dentry->d_u.d_child);
+ spin_unlock(&dcache_lock);
+ remove_dir(dentry);
+}
+
+static int rebind_subsystems(struct cgroupfs_root *root,
+         unsigned long final_bits)
```

```
+{
+ unsigned long added_bits, removed_bits;
+ struct cgroup *cont = &root->top_cgroup;
+ int i;
+
+ removed_bits = root->actual_subsys_bits & ~final_bits;
+ added_bits = final_bits & ~root->actual_subsys_bits;
+ /* Check that any added subsystems are currently free */
+ for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
+  unsigned long long bit = 1ull << i;
+  struct cgroup_subsys *ss = subsys[i];
+  if (!(bit & added_bits))
+   continue;
+  if (ss->root != &rootnode) {
+   /* Subsystem isn't free */
+   return -EBUSY;
+  }
+ }
+
+ /* Currently we don't handle adding/removing subsystems when
+  * any child cgroups exist. This is theoretically supportable
+  * but involves complex error handling, so it's being left until
+  * later */
+ if (!list_empty(&cont->children))
+  return -EBUSY;
+
+ /* Process each subsystem */
+ for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
+  struct cgroup_subsys *ss = subsys[i];
+  unsigned long bit = 1UL << i;
+  if (bit & added_bits) {
+   /* We're binding this subsystem to this hierarchy */
+   BUG_ON(cont->subsys[i]);
+   BUG_ON(!dummytop->subsys[i]);
+   BUG_ON(dummytop->subsys[i]->cgroup != dummytop);
+   cont->subsys[i] = dummytop->subsys[i];
+   cont->subsys[i]->cgroup = cont;
+   list_add(&ss->sibling, &root->subsys_list);
+   rcu_assign_pointer(ss->root, root);
+   if (ss->bind)
+    ss->bind(ss, cont);
+
+  } else if (bit & removed_bits) {
+   /* We're removing this subsystem */
+   BUG_ON(cont->subsys[i] != dummytop->subsys[i]);
+   BUG_ON(cont->subsys[i]->cgroup != cont);
+   if (ss->bind)
+    ss->bind(ss, dummytop);
```

```
+   dummytop->subsys[i]->cgroup = dummytop;
+   cont->subsys[i] = NULL;
+   rcu_assign_pointer(subsys[i]->root, &rootnode);
+   list_del(&ss->sibling);
+  } else if (bit & final_bits) {
+   /* Subsystem state should already exist */
+   BUG_ON(!cont->subsys[i]);
+  } else {
+   /* Subsystem state shouldn't exist */
+   BUG_ON(cont->subsys[i]);
+  }
+ }
+ root->subsys_bits = root->actual_subsys_bits = final_bits;
+ synchronize_rcu();
+
+ return 0;
+}
+
+static int cgroup_show_options(struct seq_file *seq, struct vfsmount *vfs)
+{
+ struct cgroupfs_root *root = vfs->mnt_sb->s_fs_info;
+ struct cgroup_subsys *ss;
+
+ mutex_lock(&cgroup_mutex);
+ for_each_subsys(root, ss)
+  seq_printf(seq, ",%s", ss->name);
+ if (test_bit(ROOT_NOPREFIX, &root->flags))
+  seq_puts(seq, ",noprefix");
+ mutex_unlock(&cgroup_mutex);
+ return 0;
+}
+
+struct cgroup_sb_opts {
+ unsigned long subsys_bits;
+ unsigned long flags;
+};
+
+/* Convert a hierarchy specifier into a bitmask of subsystems and
+ * flags. */
+static int parse_cgroupfs_options(char *data,
+         struct cgroup_sb_opts *opts)
+{
+ char *token, *o = data ?: "all";
+
+ opts->subsys_bits = 0;
+ opts->flags = 0;
+
+ while ((token = strsep(&o, ",")) != NULL) {
```

```
+  if (!*token)
+   return -EINVAL;
+  if (!strcmp(token, "all")) {
+   opts->subsys_bits = (1 << CGROUP_SUBSYS_COUNT) - 1;
+  } else if (!strcmp(token, "noprefix")) {
+   set_bit(ROOT_NOPREFIX, &opts->flags);
+  } else {
+   struct cgroup_subsys *ss;
+   int i;
+   for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
+    ss = subsys[i];
+    if (!strcmp(token, ss->name)) {
+     set_bit(i, &opts->subsys_bits);
+     break;
+    }
+   }
+   if (i == CGROUP_SUBSYS_COUNT)
+    return -ENOENT;
+  }
+ }
+
+ /* We can't have an empty hierarchy */
+ if (!opts->subsys_bits)
+  return -EINVAL;
+
+ return 0;
+}
+
+static int cgroup_remount(struct super_block *sb, int *flags, char *data)
+{
+ int ret = 0;
+ struct cgroupfs_root *root = sb->s_fs_info;
+ struct cgroup *cont = &root->top_cgroup;
+ struct cgroup_sb_opts opts;
+
+ mutex_lock(&cont->dentry->d_inode->i_mutex);
+ mutex_lock(&cgroup_mutex);
+
+ /* See what subsystems are wanted */
+ ret = parse_cgroupfs_options(data, &opts);
+ if (ret)
+  goto out_unlock;
+
+ /* Don't allow flags to change at remount */
+ if (opts.flags != root->flags) {
+  ret = -EINVAL;
+  goto out_unlock;
+ }
```

```
+
+ ret = rebind_subsystems(root, opts.subsys_bits);
+
+ /* (re)populate subsystem files */
+ if (!ret)
+  cgroup_populate_dir(cont);
+
+ out_unlock:
+ mutex_unlock(&cgroup_mutex);
+ mutex_unlock(&cont->dentry->d_inode->i_mutex);
+ return ret;
+}
+
+static struct super_operations cgroup_ops = {
+ .statfs = simple_statfs,
+ .drop_inode = generic_delete_inode,
+ .show_options = cgroup_show_options,
+ .remount_fs = cgroup_remount,
+};
+
+static void init_cgroup_root(struct cgroupfs_root *root)
+{
+ struct cgroup *cont = &root->top_cgroup;
+ INIT_LIST_HEAD(&root->subsys_list);
+ INIT_LIST_HEAD(&root->root_list);
+ root->number_of_cgroups = 1;
+ cont->root = root;
+ cont->top_cgroup = cont;
+ INIT_LIST_HEAD(&cont->sibling);
+ INIT_LIST_HEAD(&cont->children);
+}
+
+static int cgroup_test_super(struct super_block *sb, void *data)
+{
+ struct cgroupfs_root *new = data;
+ struct cgroupfs_root *root = sb->s_fs_info;
+
+ /* First check subsystems */
+ if (new->subsys_bits != root->subsys_bits)
+     return 0;
+
+ /* Next check flags */
+ if (new->flags != root->flags)
+  return 0;
+
+ return 1;
+}
+
```

```
+static int cgroup_set_super(struct super_block *sb, void *data)
+{
+ int ret;
+ struct cgroupfs_root *root = data;
+
+ ret = set_anon_super(sb, NULL);
+ if (ret)
+  return ret;
+
+ sb->s_fs_info = root;
+ root->sb = sb;
+
+ sb->s_blocksize = PAGE_CACHE_SIZE;
+ sb->s_blocksize_bits = PAGE_CACHE_SHIFT;
+ sb->s_magic = CGROUP_SUPER_MAGIC;
+ sb->s_op = &cgroup_ops;
+
+ return 0;
+}
+
+static int cgroup_get_rootdir(struct super_block *sb)
+{
+ struct inode *inode =
+  cgroup_new_inode(S_IFDIR | S_IRUGO | S_IXUGO | S_IWUSR, sb);
+ struct dentry *dentry;
+
+ if (!inode)
+  return -ENOMEM;
+
+ inode->i_op = &simple_dir_inode_operations;
+ inode->i_fop = &simple_dir_operations;
+ inode->i_op = &cgroup_dir_inode_operations;
+ /* directories start off with i_nlink == 2 (for "." entry) */
+ inc_nlink(inode);
+ dentry = d_alloc_root(inode);
+ if (!dentry) {
+  iput(inode);
+  return -ENOMEM;
+ }
+ sb->s_root = dentry;
+ return 0;
+}
+
+static int cgroup_get_sb(struct file_system_type *fs_type,
+     int flags, const char *unused_dev_name,
+     void *data, struct vfsmount *mnt)
+{
+ struct cgroup_sb_opts opts;
```

```
+ int ret = 0;
+ struct super_block *sb;
+ struct cgroupfs_root *root;
+
+ /* First find the desired set of subsystems */
+ ret = parse_cgroupfs_options(data, &opts);
+ if (ret)
+  return ret;
+
+ root = kzalloc(sizeof(*root), GFP_KERNEL);
+ if (!root)
+  return -ENOMEM;
+
+ init_cgroup_root(root);
+ root->subsys_bits = opts.subsys_bits;
+ root->flags = opts.flags;
+
+ sb = sget(fs_type, cgroup_test_super, cgroup_set_super, root);
+
+ if (IS_ERR(sb)) {
+  kfree(root);
+  return PTR_ERR(sb);
+ }
+
+ if (sb->s_fs_info != root) {
+  /* Reusing an existing superblock */
+  BUG_ON(sb->s_root == NULL);
+  kfree(root);
+  root = NULL;
+ } else {
+  /* New superblock */
+  struct cgroup *cont = &root->top_cgroup;
+
+  BUG_ON(sb->s_root != NULL);
+
+  ret = cgroup_get_rootdir(sb);
+  if (ret)
+   goto drop_new_super;
+
+  mutex_lock(&cgroup_mutex);
+
+  ret = rebind_subsystems(root, root->subsys_bits);
+  if (ret == -EBUSY) {
+   mutex_unlock(&cgroup_mutex);
+   goto drop_new_super;
+  }
+
+  /* EBUSY should be the only error here */
```

```
+  BUG_ON(ret);
+
+  list_add(&root->root_list, &roots);
+
+  sb->s_root->d_fsdata = &root->top_cgroup;
+  root->top_cgroup.dentry = sb->s_root;
+
+  BUG_ON(!list_empty(&cont->sibling));
+  BUG_ON(!list_empty(&cont->children));
+  BUG_ON(root->number_of_cgroups != 1);
+
+  /*
+   * I believe that it's safe to nest i_mutex inside
+   * cgroup_mutex in this case, since no-one else can
+   * be accessing this directory yet. But we still need
+   * to teach lockdep that this is the case - currently
+   * a cgroupfs remount triggers a lockdep warning
+   */
+  mutex_lock(&cont->dentry->d_inode->i_mutex);
+  cgroup_populate_dir(cont);
+  mutex_unlock(&cont->dentry->d_inode->i_mutex);
+  mutex_unlock(&cgroup_mutex);
+ }
+
+ return simple_set_mnt(mnt, sb);
+
+ drop_new_super:
+ up_write(&sb->s_umount);
+ deactivate_super(sb);
+ return ret;
+}
+
+static void cgroup_kill_sb(struct super_block *sb) {
+ struct cgroupfs_root *root = sb->s_fs_info;
+ struct cgroup *cont = &root->top_cgroup;
+ int ret;
+
+ BUG_ON(!root);
+
+ BUG_ON(root->number_of_cgroups != 1);
+ BUG_ON(!list_empty(&cont->children));
+ BUG_ON(!list_empty(&cont->sibling));
+
+ mutex_lock(&cgroup_mutex);
+
+ /* Rebind all subsystems back to the default hierarchy */
+ ret = rebind_subsystems(root, 0);
+ /* Shouldn't be able to fail ... */
```

```
+ BUG_ON(ret);
+
+ if (!list_empty(&root->root_list))
+  list_del(&root->root_list);
+ mutex_unlock(&cgroup_mutex);
+
+ kfree(root);
+ kill_litter_super(sb);
+}
+
+static struct file_system_type cgroup_fs_type = {
+ .name = "cgroup",
+ .get_sb = cgroup_get_sb,
+ .kill_sb = cgroup_kill_sb,
+};
+
+static inline struct cgroup *__d_cont(struct dentry *dentry)
+{
+ return dentry->d_fsdata;
+}
+
+static inline struct cftype *__d_cft(struct dentry *dentry)
+{
+ return dentry->d_fsdata;
+}
+
+/*
+ * Called with cgroup_mutex held.  Writes path of cgroup into buf.
+ * Returns 0 on success, -errno on error.
+ */
+int cgroup_path(const struct cgroup *cont, char *buf, int buflen)
+{
+ char *start;
+
+ start = buf + buflen;
+
+ *--start = '\0';
+ for (;;) {
+  int len = cont->dentry->d_name.len;
+  if ((start -= len) < buf)
+   return -ENAMETOOLONG;
+  memcpy(start, cont->dentry->d_name.name, len);
+  cont = cont->parent;
+  if (!cont)
+   break;
+  if (!cont->parent)
+   continue;
+  if (--start < buf)
```

```
+   return -ENAMETOOLONG;
+  *start = '/';
+ }
+ memmove(buf, start, buf + buflen - start);
+ return 0;
+}
+
+/* The various types of files and directories in a cgroup file system */
+
+enum cgroup_filetype {
+ FILE_ROOT,
+ FILE_DIR,
+ FILE_TASKLIST,
+};
+
+static ssize_t cgroup_file_write(struct file *file, const char __user *buf,
+      size_t nbytes, loff_t *ppos)
+{
+ struct cftype *cft = __d_cft(file->f_dentry);
+ struct cgroup *cont = __d_cont(file->f_dentry->d_parent);
+
+ if (!cft)
+  return -ENODEV;
+ if (!cft->write)
+  return -EINVAL;
+
+ return cft->write(cont, cft, file, buf, nbytes, ppos);
+}
+
+static ssize_t cgroup_read_uint(struct cgroup *cont, struct cftype *cft,
+       struct file *file,
+       char __user *buf, size_t nbytes,
+       loff_t *ppos)
+{
+ char tmp[64];
+ u64 val = cft->read_uint(cont, cft);
+ int len = sprintf(tmp, "%llu\n", (unsigned long long) val);
+
+ return simple_read_from_buffer(buf, nbytes, ppos, tmp, len);
+}
+
+static ssize_t cgroup_file_read(struct file *file, char __user *buf,
+       size_t nbytes, loff_t *ppos)
+{
+ struct cftype *cft = __d_cft(file->f_dentry);
+ struct cgroup *cont = __d_cont(file->f_dentry->d_parent);
+
+ if (!cft)
```

```
+  return -ENODEV;
+
+ if (cft->read)
+  return cft->read(cont, cft, file, buf, nbytes, ppos);
+ if (cft->read_uint)
+  return cgroup_read_uint(cont, cft, file, buf, nbytes, ppos);
+ return -EINVAL;
+}
+
+static int cgroup_file_open(struct inode *inode, struct file *file)
+{
+ int err;
+ struct cftype *cft;
+
+ err = generic_file_open(inode, file);
+ if (err)
+  return err;
+
+ cft = __d_cft(file->f_dentry);
+ if (!cft)
+  return -ENODEV;
+ if (cft->open)
+  err = cft->open(inode, file);
+ else
+  err = 0;
+
+ return err;
+}
+
+static int cgroup_file_release(struct inode *inode, struct file *file)
+{
+ struct cftype *cft = __d_cft(file->f_dentry);
+ if (cft->release)
+  return cft->release(inode, file);
+ return 0;
+}
+
+/*
+ * cgroup_rename - Only allow simple rename of directories in place.
+ */
+static int cgroup_rename(struct inode *old_dir, struct dentry *old_dentry,
+       struct inode *new_dir, struct dentry *new_dentry)
+{
+ if (!S_ISDIR(old_dentry->d_inode->i_mode))
+  return -ENOTDIR;
+ if (new_dentry->d_inode)
+  return -EEXIST;
+ if (old_dir != new_dir)
```

```
+  return -EIO;
+ return simple_rename(old_dir, old_dentry, new_dir, new_dentry);
+}
+
+static struct file_operations cgroup_file_operations = {
+ .read = cgroup_file_read,
+ .write = cgroup_file_write,
+ .llseek = generic_file_llseek,
+ .open = cgroup_file_open,
+ .release = cgroup_file_release,
+};
+
+static struct inode_operations cgroup_dir_inode_operations = {
+ .lookup = simple_lookup,
+ .mkdir = cgroup_mkdir,
+ .rmdir = cgroup_rmdir,
+ .rename = cgroup_rename,
+};
+
+static int cgroup_create_file(struct dentry *dentry, int mode,
+    struct super_block *sb)
+{
+ struct inode *inode;
+
+ if (!dentry)
+  return -ENOENT;
+ if (dentry->d_inode)
+  return -EEXIST;
+
+ inode = cgroup_new_inode(mode, sb);
+ if (!inode)
+  return -ENOMEM;
+
+ if (S_ISDIR(mode)) {
+  inode->i_op = &cgroup_dir_inode_operations;
+  inode->i_fop = &simple_dir_operations;
+
+  /* start off with i_nlink == 2 (for "." entry) */
+  inc_nlink(inode);
+
+  /* start with the directory inode held, so that we can
+   * populate it without racing with another mkdir */
+  mutex_lock(&inode->i_mutex);
+ } else if (S_ISREG(mode)) {
+  inode->i_size = 0;
+  inode->i_fop = &cgroup_file_operations;
+ }
+
```

```
+ d_instantiate(dentry, inode);
+ dget(dentry); /* Extra count - pin the dentry in core */
+ return 0;
+}
+
+/*
+ * cgroup_create_dir - create a directory for an object.
+ * cont: the cgroup we create the directory for.
+ *  It must have a valid ->parent field
+ *  And we are going to fill its ->dentry field.
+ * name: The name to give to the cgroup directory. Will be copied.
+ * mode: mode to set on new directory.
+ */
+static int cgroup_create_dir(struct cgroup *cont, struct dentry *dentry,
+    int mode)
+{
+ struct dentry *parent;
+ int error = 0;
+
+ parent = cont->parent->dentry;
+ if (IS_ERR(dentry))
+  return PTR_ERR(dentry);
+ error = cgroup_create_file(dentry, S_IFDIR | mode, cont->root->sb);
+ if (!error) {
+  dentry->d_fsdata = cont;
+  inc_nlink(parent->d_inode);
+  cont->dentry = dentry;
+ }
+ dput(dentry);
+
+ return error;
+}
+
+int cgroup_add_file(struct cgroup *cont,
+         struct cgroup_subsys *subsys,
+         const struct cftype *cft)
+{
+ struct dentry *dir = cont->dentry;
+ struct dentry *dentry;
+ int error;
+
+ char name[MAX_CGROUP_TYPE_NAMELEN + MAX_CFTYPE_NAME + 2] = { 0 };
+ if (subsys && !test_bit(ROOT_NOPREFIX, &cont->root->flags)) {
+  strcpy(name, subsys->name);
+  strcat(name, ".");
+ }
+ strcat(name, cft->name);
+ BUG_ON(!mutex_is_locked(&dir->d_inode->i_mutex));
```

```
+ dentry = cgroup_get_dentry(dir, name);
+ if (!IS_ERR(dentry)) {
+  error = cgroup_create_file(dentry, 0644 | S_IFREG,
+      cont->root->sb);
+  if (!error)
+   dentry->d_fsdata = (void *)cft;
+  dput(dentry);
+ } else
+  error = PTR_ERR(dentry);
+ return error;
+}
+
+int cgroup_add_files(struct cgroup *cont,
+   struct cgroup_subsys *subsys,
+   const struct cftype cft[],
+   int count)
+{
+ int i, err;
+ for (i = 0; i < count; i++) {
+  err = cgroup_add_file(cont, subsys, &cft[i]);
+  if (err)
+   return err;
+ }
+ return 0;
+}
+
+static int cgroup_populate_dir(struct cgroup *cont)
+{
+ int err;
+ struct cgroup_subsys *ss;
+
+ /* First clear out any existing files */
+ cgroup_clear_directory(cont->dentry);
+
+ for_each_subsys(cont->root, ss) {
+  if (ss->populate && (err = ss->populate(ss, cont)) < 0)
+   return err;
+ }
+
+ return 0;
+}
+
+static void init_cgroup_css(struct cgroup_subsys_state *css,
+         struct cgroup_subsys *ss,
+         struct cgroup *cont)
+{
+ css->cgroup = cont;
+ atomic_set(&css->refcnt, 0);
```

```
+ css->flags = 0;
+ if (cont == dummytop)
+  set_bit(CSS_ROOT, &css->flags);
+ BUG_ON(cont->subsys[ss->subsys_id]);
+ cont->subsys[ss->subsys_id] = css;
+}
+
+/*
+ * cgroup_create - create a cgroup
+ * parent: cgroup that will be parent of the new cgroup.
+ * name:  name of the new cgroup. Will be strcpy'ed.
+ * mode:  mode to set on new inode
+ *
+ * Must be called with the mutex on the parent inode held
+ */
+
+static long cgroup_create(struct cgroup *parent, struct dentry *dentry,
+        int mode)
+{
+ struct cgroup *cont;
+ struct cgroupfs_root *root = parent->root;
+ int err = 0;
+ struct cgroup_subsys *ss;
+ struct super_block *sb = root->sb;
+
+ cont = kzalloc(sizeof(*cont), GFP_KERNEL);
+ if (!cont)
+  return -ENOMEM;
+
+ /* Grab a reference on the superblock so the hierarchy doesn't
+  * get deleted on unmount if there are child cgroups.  This
+  * can be done outside cgroup_mutex, since the sb can't
+  * disappear while someone has an open control file on the
+  * fs */
+ atomic_inc(&sb->s_active);
+
+ mutex_lock(&cgroup_mutex);
+
+ cont->flags = 0;
+ INIT_LIST_HEAD(&cont->sibling);
+ INIT_LIST_HEAD(&cont->children);
+
+ cont->parent = parent;
+ cont->root = parent->root;
+ cont->top_cgroup = parent->top_cgroup;
+
+ for_each_subsys(root, ss) {
+  struct cgroup_subsys_state *css = ss->create(ss, cont);
```

```
+  if (IS_ERR(css)) {
+   err = PTR_ERR(css);
+   goto err_destroy;
+  }
+  init_cgroup_css(css, ss, cont);
+ }
+
+ list_add(&cont->sibling, &cont->parent->children);
+ root->number_of_cgroups++;
+
+ err = cgroup_create_dir(cont, dentry, mode);
+ if (err < 0)
+  goto err_remove;
+
+ /* The cgroup directory was pre-locked for us */
+ BUG_ON(!mutex_is_locked(&cont->dentry->d_inode->i_mutex));
+
+ err = cgroup_populate_dir(cont);
+ /* If err < 0, we have a half-filled directory - oh well ;) */
+
+ mutex_unlock(&cgroup_mutex);
+ mutex_unlock(&cont->dentry->d_inode->i_mutex);
+
+ return 0;
+
+ err_remove:
+
+ list_del(&cont->sibling);
+ root->number_of_cgroups--;
+
+ err_destroy:
+
+ for_each_subsys(root, ss) {
+  if (cont->subsys[ss->subsys_id])
+   ss->destroy(ss, cont);
+ }
+
+ mutex_unlock(&cgroup_mutex);
+
+ /* Release the reference count that we took on the superblock */
+ deactivate_super(sb);
+
+ kfree(cont);
+ return err;
+}
+
+static int cgroup_mkdir(struct inode *dir, struct dentry *dentry, int mode)
+{
```

```
+ struct cgroup *c_parent = dentry->d_parent->d_fsdata;
+
+ /* the vfs holds inode->i_mutex already */
+ return cgroup_create(c_parent, dentry, mode | S_IFDIR);
+}
+
+static int cgroup_rmdir(struct inode *unused_dir, struct dentry *dentry)
+{
+ struct cgroup *cont = dentry->d_fsdata;
+ struct dentry *d;
+ struct cgroup *parent;
+ struct cgroup_subsys *ss;
+ struct super_block *sb;
+ struct cgroupfs_root *root;
+ int css_busy = 0;
+
+ /* the vfs holds both inode->i_mutex already */
+
+ mutex_lock(&cgroup_mutex);
+ if (atomic_read(&cont->count) != 0) {
+  mutex_unlock(&cgroup_mutex);
+  return -EBUSY;
+ }
+ if (!list_empty(&cont->children)) {
+  mutex_unlock(&cgroup_mutex);
+  return -EBUSY;
+ }
+
+ parent = cont->parent;
+ root = cont->root;
+ sb = root->sb;
+
+ /* Check the reference count on each subsystem. Since we
+  * already established that there are no tasks in the
+  * cgroup, if the css refcount is also 0, then there should
+  * be no outstanding references, so the subsystem is safe to
+  * destroy */
+ for_each_subsys(root, ss) {
+  struct cgroup_subsys_state *css;
+  css = cont->subsys[ss->subsys_id];
+  if (atomic_read(&css->refcnt)) {
+   css_busy = 1;
+   break;
+  }
+ }
+ if (css_busy) {
+  mutex_unlock(&cgroup_mutex);
+  return -EBUSY;
```

```
+ }
+
+ for_each_subsys(root, ss) {
+  if (cont->subsys[ss->subsys_id])
+   ss->destroy(ss, cont);
+ }
+
+ set_bit(CONT_REMOVED, &cont->flags);
+ /* delete my sibling from parent->children */
+ list_del(&cont->sibling);
+ spin_lock(&cont->dentry->d_lock);
+ d = dget(cont->dentry);
+ cont->dentry = NULL;
+ spin_unlock(&d->d_lock);
+
+ cgroup_d_remove_dir(d);
+ dput(d);
+ root->number_of_cgroups--;
+
+ mutex_unlock(&cgroup_mutex);
+ /* Drop the active superblock reference that we took when we
+  * created the cgroup */
+ deactivate_super(sb);
+ return 0;
+}
+
+static void cgroup_init_subsys(struct cgroup_subsys *ss)
+{
+ struct task_struct *g, *p;
+ struct cgroup_subsys_state *css;
+ printk(KERN_ERR "Initializing cgroup subsys %s\n", ss->name);
+
+ /* Create the top cgroup state for this subsystem */
+ ss->root = &rootnode;
+ css = ss->create(ss, dummytop);
+ /* We don't handle early failures gracefully */
+ BUG_ON(IS_ERR(css));
+ init_cgroup_css(css, ss, dummytop);
+
+ /* Update all tasks to contain a subsys pointer to this state
+  * - since the subsystem is newly registered, all tasks are in
+  * the subsystem's top cgroup. */
+
+  /* If this subsystem requested that it be notified with fork
+   * events, we should send it one now for every process in the
+   * system */
+
+ read_lock(&tasklist_lock);
```

```
+ init_task.cgroups.subsys[ss->subsys_id] = css;
+ if (ss->fork)
+  ss->fork(ss, &init_task);
+
+ do_each_thread(g, p) {
+  printk(KERN_INFO "Setting task %p css to %p (%d)\n", css, p, p->pid);
+  p->cgroups.subsys[ss->subsys_id] = css;
+  if (ss->fork)
+   ss->fork(ss, p);
+ } while_each_thread(g, p);
+ read_unlock(&tasklist_lock);
+
+ need_forkexit_callback |= ss->fork || ss->exit;
+
+ ss->active = 1;
+}
+
+/**
+ * cgroup_init_early - initialize cgroups at system boot, and
+ * initialize any subsystems that request early init.
+ */
+int __init cgroup_init_early(void)
+{
+ int i;
+ init_cgroup_root(&rootnode);
+ list_add(&rootnode.root_list, &roots);
+
+ for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
+  struct cgroup_subsys *ss = subsys[i];
+
+  BUG_ON(!ss->name);
+  BUG_ON(strlen(ss->name) > MAX_CGROUP_TYPE_NAMELEN);
+  BUG_ON(!ss->create);
+  BUG_ON(!ss->destroy);
+  if (ss->subsys_id != i) {
+   printk(KERN_ERR "Subsys %s id == %d\n",
+          ss->name, ss->subsys_id);
+   BUG();
+  }
+
+  if (ss->early_init)
+   cgroup_init_subsys(ss);
+ }
+ return 0;
+}
+
+/**
+ * cgroup_init - register cgroup filesystem and /proc file, and
```

```
+ * initialize any subsystems that didn't request early init.
+ */
+int __init cgroup_init(void)
+{
+ int err;
+ int i;
+
+ for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
+  struct cgroup_subsys *ss = subsys[i];
+  if (!ss->early_init)
+   cgroup_init_subsys(ss);
+ }
+
+ err = register_filesystem(&cgroup_fs_type);
+ if (err < 0)
+  goto out;
+
+out:
+ return err;
+}
```

_

--

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 02/29] task containersv11 basic task container framework fix

Posted by Paul Menage on Tue, 11 Sep 2007 19:52:41 GMT

View Forum Message <> Reply to Message

From: "Paul Menage" <menage@google.com>

On 7/28/07, Andrew Morton <akpm@linux-foundation.org> wrote:

```
> >  [<c0154ad2>] proc_cpuset_show+0x5e/0xb9
> >  [<c01976b1>] seq_read+0xef/0x266
> >  [<c0181398>] vfs_read+0x8e/0x117
> >  [<c01817c9>] sys_read+0x3d/0x61
> >  [<c010401e>] sysenter_past_esp+0x5f/0x99
> >  =======================
> > INFO: lockdep is turned off.
> > Code: 00 89 d8 83 c4 0c 5b 5e 5f 5d c3 55 89 e5 57 56 53 83 ec 08 89
> > 45 f0 89 55 ec 89 d3 01 cb 8d 43 ff c6 43 ff 00 8b 55 f0 8b 72 1c <8b>
> > 56 3c 29 d0 3b 45 ec 72 45 89 d1 c1 e9 02 8b 76 40 89 c7 f3
> > EIP: [<c01536a7>] cgroup_path+0x20/0x7b SS:ESP 0068:c5ccff00
>
```

Oops, this is a case of trying to read /proc/self/cpuset when cpusets
are completely unmounted. It works fine when cpusets is mounted as
part of any hierarchy, or part of an active but unmounted hierarchy.
Attached patch fixes it, and should be rolled in to the first patch of
the series. (basic framework patch).

Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

 kernel/cgroup.c | 9 +++++++++
 1 file changed, 9 insertions(+)

diff -puN kernel/cgroup.c~task-cgroupsv11-basic-task-cgroup-framework-fix kernel/cgroup.c
--- a/kernel/cgroup.c~task-cgroupsv11-basic-task-cgroup-framework-fix
+++ a/kernel/cgroup.c
@@ -683,6 +683,15 @@ int cgroup_path(const struct containe
 {
  char *start;

+ if (cont == dummytop) {
+ /*
+   * Inactive subsystems have no dentry for their root
+   * cgroup
+   */
+ strcpy(buf, "/");
+ return 0;
+ }
+
  start = buf + buflen;

  *--start = '\0';
_

--

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 03/29] task containersv11 add tasks file interface
Posted by Paul Menage on Tue, 11 Sep 2007 19:52:42 GMT
View Forum Message <> Reply to Message

From: Paul Menage <menage@google.com>

Add the per-directory "tasks" file for cgroupfs mounts; this allows the
user to determine which tasks are members of a cgroup by reading a
cgroup's "tasks", and to move a task into a cgroup by writing its pid to
its "tasks".

Signed-off-by: Paul Menage <menage@google.com>
Cc: Serge E. Hallyn <serue@us.ibm.com>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Dave Hansen <haveblue@us.ibm.com>
Cc: Balbir Singh <balbir@in.ibm.com>
Cc: Paul Jackson <pj@sgi.com>
Cc: Kirill Korotaev <dev@openvz.org>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: Srivatsa Vaddagiri <vatsa@in.ibm.com>
Cc: Cedric Le Goater <clg@fr.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

 include/linux/cgroup.h | 10 +
 kernel/cgroup.c | 359 +++-
 2 files changed, 368 insertions(+), 1 deletion(-)

diff -puN include/linux/cgroup.h~task-cgroupsv11-add-tasks-file-interface include/linux/cgroup.h
--- a/include/linux/cgroup.h~task-cgroupsv11-add-tasks-file-interface
+++ a/include/linux/cgroup.h
@@ -144,6 +144,16 @@ int cgroup_is_removed(const struct co

 int cgroup_path(const struct cgroup *cont, char *buf, int buflen);

+int __cgroup_task_count(const struct cgroup *cont);
+static inline int cgroup_task_count(const struct cgroup *cont)
+{
+ int task_count;
+ rcu_read_lock();
+ task_count = __cgroup_task_count(cont);
+ rcu_read_unlock();
+ return task_count;
+}
+
 /* Return true if the cgroup is a descendant of the current cgroup */
 int cgroup_is_descendant(const struct cgroup *cont);

diff -puN kernel/cgroup.c~task-cgroupsv11-add-tasks-file-interface kernel/cgroup.c
--- a/kernel/cgroup.c~task-cgroupsv11-add-tasks-file-interface
+++ a/kernel/cgroup.c
@@ -40,7 +40,7 @@
 #include <linux/magic.h>
 #include <linux/spinlock.h>

```
 #include <linux/string.h>
-
+#include <linux/sort.h>
 #include <asm/atomic.h>

 /* Generate an array of cgroup subsystem pointers */
@@ -713,6 +713,127 @@ int cgroup_path(const struct containe
  return 0;
 }

+/*
+ * Return the first subsystem attached to a cgroup's hierarchy, and
+ * its subsystem id.
+ */
+
+static void get_first_subsys(const struct cgroup *cont,
+   struct cgroup_subsys_state **css, int *subsys_id)
+{
+ const struct cgroupfs_root *root = cont->root;
+ const struct cgroup_subsys *test_ss;
+ BUG_ON(list_empty(&root->subsys_list));
+ test_ss = list_entry(root->subsys_list.next,
+        struct cgroup_subsys, sibling);
+ if (css) {
+  *css = cont->subsys[test_ss->subsys_id];
+  BUG_ON(!*css);
+ }
+ if (subsys_id)
+  *subsys_id = test_ss->subsys_id;
+}
+
+/*
+ * Attach task 'tsk' to cgroup 'cont'
+ *
+ * Call holding cgroup_mutex.  May take task_lock of
+ * the task 'pid' during call.
+ */
+static int attach_task(struct cgroup *cont, struct task_struct *tsk)
+{
+ int retval = 0;
+ struct cgroup_subsys *ss;
+ struct cgroup *oldcont;
+ struct css_set *cg = &tsk->cgroups;
+ struct cgroupfs_root *root = cont->root;
+ int i;
+ int subsys_id;
+
+ get_first_subsys(cont, NULL, &subsys_id);
```

```
+
+ /* Nothing to do if the task is already in that cgroup */
+ oldcont = task_cgroup(tsk, subsys_id);
+ if (cont == oldcont)
+  return 0;
+
+ for_each_subsys(root, ss) {
+  if (ss->can_attach) {
+   retval = ss->can_attach(ss, cont, tsk);
+   if (retval) {
+    return retval;
+   }
+  }
+ }
+
+ task_lock(tsk);
+ if (tsk->flags & PF_EXITING) {
+  task_unlock(tsk);
+  return -ESRCH;
+ }
+ /* Update the css_set pointers for the subsystems in this
+  * hierarchy */
+ for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
+  if (root->subsys_bits & (1ull << i)) {
+   /* Subsystem is in this hierarchy. So we want
+    * the subsystem state from the new
+    * cgroup. Transfer the refcount from the
+    * old to the new */
+   atomic_inc(&cont->count);
+   atomic_dec(&cg->subsys[i]->cgroup->count);
+   rcu_assign_pointer(cg->subsys[i], cont->subsys[i]);
+  }
+ }
+ task_unlock(tsk);
+
+ for_each_subsys(root, ss) {
+  if (ss->attach) {
+   ss->attach(ss, cont, oldcont, tsk);
+  }
+ }
+
+ synchronize_rcu();
+ return 0;
+}
+
+/*
+ * Attach task with pid 'pid' to cgroup 'cont'. Call with
+ * cgroup_mutex, may take task_lock of task
```

```
+ */
+static int attach_task_by_pid(struct cgroup *cont, char *pidbuf)
+{
+ pid_t pid;
+ struct task_struct *tsk;
+ int ret;
+
+ if (sscanf(pidbuf, "%d", &pid) != 1)
+  return -EIO;
+
+ if (pid) {
+  rcu_read_lock();
+  tsk = find_task_by_pid(pid);
+  if (!tsk || tsk->flags & PF_EXITING) {
+   rcu_read_unlock();
+   return -ESRCH;
+  }
+  get_task_struct(tsk);
+  rcu_read_unlock();
+
+  if ((current->euid) && (current->euid != tsk->uid)
+      && (current->euid != tsk->suid)) {
+   put_task_struct(tsk);
+   return -EACCES;
+  }
+ } else {
+  tsk = current;
+  get_task_struct(tsk);
+ }
+
+ ret = attach_task(cont, tsk);
+ put_task_struct(tsk);
+ return ret;
+}
+
 /* The various types of files and directories in a cgroup file system */

 enum cgroup_filetype {
@@ -721,6 +842,55 @@ enum cgroup_filetype {
  FILE_TASKLIST,
 };

+static ssize_t cgroup_common_file_write(struct cgroup *cont,
+        struct cftype *cft,
+        struct file *file,
+        const char __user *userbuf,
+        size_t nbytes, loff_t *unused_ppos)
+{
```

```
+ enum cgroup_filetype type = cft->private;
+ char *buffer;
+ int retval = 0;
+
+ if (nbytes >= PATH_MAX)
+  return -E2BIG;
+
+ /* +1 for nul-terminator */
+ buffer = kmalloc(nbytes + 1, GFP_KERNEL);
+ if (buffer == NULL)
+  return -ENOMEM;
+
+ if (copy_from_user(buffer, userbuf, nbytes)) {
+  retval = -EFAULT;
+  goto out1;
+ }
+ buffer[nbytes] = 0; /* nul-terminate */
+
+ mutex_lock(&cgroup_mutex);
+
+ if (cgroup_is_removed(cont)) {
+  retval = -ENODEV;
+  goto out2;
+ }
+
+ switch (type) {
+ case FILE_TASKLIST:
+  retval = attach_task_by_pid(cont, buffer);
+  break;
+ default:
+  retval = -EINVAL;
+  goto out2;
+ }
+
+ if (retval == 0)
+  retval = nbytes;
+out2:
+ mutex_unlock(&cgroup_mutex);
+out1:
+ kfree(buffer);
+ return retval;
+}
+
 static ssize_t cgroup_file_write(struct file *file, const char __user *buf,
       size_t nbytes, loff_t *ppos)
 {
@@ -924,6 +1094,189 @@ int cgroup_add_files(struct cgroup
  return 0;
```

```
 }

+/* Count the number of tasks in a cgroup. Could be made more
+ * time-efficient but less space-efficient with more linked lists
+ * running through each cgroup and the css_set structures that
+ * referenced it. Must be called with tasklist_lock held for read or
+ * write or in an rcu critical section.
+ */
+int __cgroup_task_count(const struct cgroup *cont)
+{
+ int count = 0;
+ struct task_struct *g, *p;
+ struct cgroup_subsys_state *css;
+ int subsys_id;
+
+ get_first_subsys(cont, &css, &subsys_id);
+ do_each_thread(g, p) {
+  if (task_subsys_state(p, subsys_id) == css)
+   count ++;
+ } while_each_thread(g, p);
+ return count;
+}
+
+/*
+ * Stuff for reading the 'tasks' file.
+ *
+ * Reading this file can return large amounts of data if a cgroup has
+ * *lots* of attached tasks. So it may need several calls to read(),
+ * but we cannot guarantee that the information we produce is correct
+ * unless we produce it entirely atomically.
+ *
+ * Upon tasks file open(), a struct ctr_struct is allocated, that
+ * will have a pointer to an array (also allocated here).  The struct
+ * ctr_struct * is stored in file->private_data.  Its resources will
+ * be freed by release() when the file is closed.  The array is used
+ * to sprintf the PIDs and then used by read().
+ */
+struct ctr_struct {
+ char *buf;
+ int bufsz;
+};
+
+/*
+ * Load into 'pidarray' up to 'npids' of the tasks using cgroup
+ * 'cont'.  Return actual number of pids loaded.  No need to
+ * task_lock(p) when reading out p->cgroup, since we're in an RCU
+ * read section, so the css_set can't go away, and is
+ * immutable after creation.
```

```
+ */
+static int pid_array_load(pid_t *pidarray, int npids, struct cgroup *cont)
+{
+ int n = 0;
+ struct task_struct *g, *p;
+ struct cgroup_subsys_state *css;
+ int subsys_id;
+
+ get_first_subsys(cont, &css, &subsys_id);
+ rcu_read_lock();
+ do_each_thread(g, p) {
+  if (task_subsys_state(p, subsys_id) == css) {
+   pidarray[n++] = pid_nr(task_pid(p));
+   if (unlikely(n == npids))
+    goto array_full;
+  }
+ } while_each_thread(g, p);
+
+array_full:
+ rcu_read_unlock();
+ return n;
+}
+
+static int cmppid(const void *a, const void *b)
+{
+ return *(pid_t *)a - *(pid_t *)b;
+}
+
+/*
+ * Convert array 'a' of 'npids' pid_t's to a string of newline separated
+ * decimal pids in 'buf'.  Don't write more than 'sz' chars, but return
+ * count 'cnt' of how many chars would be written if buf were large enough.
+ */
+static int pid_array_to_buf(char *buf, int sz, pid_t *a, int npids)
+{
+ int cnt = 0;
+ int i;
+
+ for (i = 0; i < npids; i++)
+  cnt += snprintf(buf + cnt, max(sz - cnt, 0), "%d\n", a[i]);
+ return cnt;
+}
+
+/*
+ * Handle an open on 'tasks' file.  Prepare a buffer listing the
+ * process id's of tasks currently attached to the cgroup being opened.
+ *
+ * Does not require any specific cgroup mutexes, and does not take any.
```

```
+ */
+static int cgroup_tasks_open(struct inode *unused, struct file *file)
+{
+ struct cgroup *cont = __d_cont(file->f_dentry->d_parent);
+ struct ctr_struct *ctr;
+ pid_t *pidarray;
+ int npids;
+ char c;
+
+ if (!(file->f_mode & FMODE_READ))
+  return 0;
+
+ ctr = kmalloc(sizeof(*ctr), GFP_KERNEL);
+ if (!ctr)
+  goto err0;
+
+ /*
+  * If cgroup gets more users after we read count, we won't have
+  * enough space - tough.  This race is indistinguishable to the
+  * caller from the case that the additional cgroup users didn't
+  * show up until sometime later on.
+  */
+ npids = cgroup_task_count(cont);
+ if (npids) {
+  pidarray = kmalloc(npids * sizeof(pid_t), GFP_KERNEL);
+  if (!pidarray)
+   goto err1;
+
+  npids = pid_array_load(pidarray, npids, cont);
+  sort(pidarray, npids, sizeof(pid_t), cmppid, NULL);
+
+  /* Call pid_array_to_buf() twice, first just to get bufsz */
+  ctr->bufsz = pid_array_to_buf(&c, sizeof(c), pidarray, npids) + 1;
+  ctr->buf = kmalloc(ctr->bufsz, GFP_KERNEL);
+  if (!ctr->buf)
+   goto err2;
+  ctr->bufsz = pid_array_to_buf(ctr->buf, ctr->bufsz, pidarray, npids);
+
+  kfree(pidarray);
+ } else {
+  ctr->buf = 0;
+  ctr->bufsz = 0;
+ }
+ file->private_data = ctr;
+ return 0;
+
+err2:
+ kfree(pidarray);
```

```
+err1:
+ kfree(ctr);
+err0:
+ return -ENOMEM;
+}
+
+static ssize_t cgroup_tasks_read(struct cgroup *cont,
+        struct cftype *cft,
+        struct file *file, char __user *buf,
+        size_t nbytes, loff_t *ppos)
+{
+ struct ctr_struct *ctr = file->private_data;
+
+ return simple_read_from_buffer(buf, nbytes, ppos, ctr->buf, ctr->bufsz);
+}
+
+static int cgroup_tasks_release(struct inode *unused_inode,
+     struct file *file)
+{
+ struct ctr_struct *ctr;
+
+ if (file->f_mode & FMODE_READ) {
+  ctr = file->private_data;
+  kfree(ctr->buf);
+  kfree(ctr);
+ }
+ return 0;
+}
+
+/*
+ * for the common functions, 'private' gives the type of file
+ */
+static struct cftype cft_tasks = {
+ .name = "tasks",
+ .open = cgroup_tasks_open,
+ .read = cgroup_tasks_read,
+ .write = cgroup_common_file_write,
+ .release = cgroup_tasks_release,
+ .private = FILE_TASKLIST,
+};
+
 static int cgroup_populate_dir(struct cgroup *cont)
 {
  int err;
@@ -932,6 +1285,10 @@ static int cgroup_populate_dir(struct
  /* First clear out any existing files */
  cgroup_clear_directory(cont->dentry);
```

```
+ err = cgroup_add_file(cont, NULL, &cft_tasks);
+ if (err < 0)
+  return err;
+
  for_each_subsys(cont->root, ss) {
   if (ss->populate && (err = ss->populate(ss, cont)) < 0)
    return err;
_

--
```

Subject: [PATCH 04/29] task containersv11 add fork exit hooks
Posted by Paul Menage on Tue, 11 Sep 2007 19:52:43 GMT
View Forum Message <> Reply to Message

From: Paul Menage <menage@google.com>

tAdd adds the necessary hooks to the fork() and exit() paths to ensure that
new children inherit their parent's cgroup assignments, and that exiting
processes release reference counts on their cgroups.

Signed-off-by: Paul Menage <menage@google.com>
Cc: Serge E. Hallyn <serue@us.ibm.com>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Dave Hansen <haveblue@us.ibm.com>
Cc: Balbir Singh <balbir@in.ibm.com>
Cc: Paul Jackson <pj@sgi.com>
Cc: Kirill Korotaev <dev@openvz.org>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: Srivatsa Vaddagiri <vatsa@in.ibm.com>
Cc: Cedric Le Goater <clg@fr.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>

```
 include/linux/cgroup.h |    6 +
 kernel/cgroup.c        |  121 ++++++++++++++++++++++++++++++++++++++++++++++
 kernel/exit.c          |    2
 kernel/fork.c          |   14 +++-
 4 files changed, 141 insertions(+), 2 deletions(-)
```

diff -puN include/linux/cgroup.h~task-cgroupsv11-add-fork-exit-hooks include/linux/cgroup.h
--- a/include/linux/cgroup.h~task-cgroupsv11-add-fork-exit-hooks
+++ a/include/linux/cgroup.h

```
@@ -25,6 +25,9 @@ extern int cgroup_init(void);
 extern void cgroup_init_smp(void);
 extern void cgroup_lock(void);
 extern void cgroup_unlock(void);
+extern void cgroup_fork(struct task_struct *p);
+extern void cgroup_fork_callbacks(struct task_struct *p);
+extern void cgroup_exit(struct task_struct *p, int run_callbacks);

 /* Per-subsystem/per-cgroup state maintained by the system. */
 struct cgroup_subsys_state {
@@ -215,6 +218,9 @@ int cgroup_path(const struct containe
 static inline int cgroup_init_early(void) { return 0; }
 static inline int cgroup_init(void) { return 0; }
 static inline void cgroup_init_smp(void) {}
+static inline void cgroup_fork(struct task_struct *p) {}
+static inline void cgroup_fork_callbacks(struct task_struct *p) {}
+static inline void cgroup_exit(struct task_struct *p, int callbacks) {}

 static inline void cgroup_lock(void) {}
 static inline void cgroup_unlock(void) {}
diff -puN kernel/cgroup.c~task-cgroupsv11-add-fork-exit-hooks kernel/cgroup.c
--- a/kernel/cgroup.c~task-cgroupsv11-add-fork-exit-hooks
+++ a/kernel/cgroup.c
@@ -132,6 +132,33 @@ list_for_each_entry(_ss, &_root->subsys_
 #define for_each_root(_root) \
 list_for_each_entry(_root, &roots, root_list)

+/* Each task_struct has an embedded css_set, so the get/put
+ * operation simply takes a reference count on all the cgroups
+ * referenced by subsystems in this css_set. This can end up
+ * multiple-counting some cgroups, but that's OK - the ref-count is
+ * just a busy/not-busy indicator; ensuring that we only count each
+ * cgroup once would require taking a global lock to ensure that no
+ * subsystems moved between hierarchies while we were doing so.
+ *
+ * Possible TODO: decide at boot time based on the number of
+ * registered subsystems and the number of CPUs or NUMA nodes whether
+ * it's better for performance to ref-count every subsystem, or to
+ * take a global lock and only add one ref count to each hierarchy.
+ */
+static void get_css_set(struct css_set *cg)
+{
+ int i;
+ for (i = 0; i < CGROUP_SUBSYS_COUNT; i++)
+  atomic_inc(&cg->subsys[i]->cgroup->count);
+}
+
+static void put_css_set(struct css_set *cg)
```

```
+{
+ int i;
+ for (i = 0; i < CGROUP_SUBSYS_COUNT; i++)
+  atomic_dec(&cg->subsys[i]->cgroup->count);
+}
+
 /*
  * There is one global cgroup mutex. We also require taking
  * task_lock() when dereferencing a task's cgroup subsys pointers.
@@ -1563,3 +1590,97 @@ int __init cgroup_init(void)
 out:
  return err;
 }
+
+/**
+ * cgroup_fork - attach newly forked task to its parents cgroup.
+ * @tsk: pointer to task_struct of forking parent process.
+ *
+ * Description: A task inherits its parent's cgroup at fork().
+ *
+ * A pointer to the shared css_set was automatically copied in
+ * fork.c by dup_task_struct().  However, we ignore that copy, since
+ * it was not made under the protection of RCU or cgroup_mutex, so
+ * might no longer be a valid cgroup pointer.  attach_task() might
+ * have already changed current->cgroup, allowing the previously
+ * referenced cgroup to be removed and freed.
+ *
+ * At the point that cgroup_fork() is called, 'current' is the parent
+ * task, and the passed argument 'child' points to the child task.
+ */
+void cgroup_fork(struct task_struct *child)
+{
+ rcu_read_lock();
+ child->cgroups = rcu_dereference(current->cgroups);
+ get_css_set(&child->cgroups);
+ rcu_read_unlock();
+}
+
+/**
+ * cgroup_fork_callbacks - called on a new task very soon before
+ * adding it to the tasklist. No need to take any locks since no-one
+ * can be operating on this task
+ */
+void cgroup_fork_callbacks(struct task_struct *child)
+{
+ if (need_forkexit_callback) {
+  int i;
+  for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
```

```
+   struct cgroup_subsys *ss = subsys[i];
+   if (ss->fork)
+    ss->fork(ss, child);
+  }
+ }
+}
+
+/**
+ * cgroup_exit - detach cgroup from exiting task
+ * @tsk: pointer to task_struct of exiting process
+ *
+ * Description: Detach cgroup from @tsk and release it.
+ *
+ * Note that cgroups marked notify_on_release force every task in
+ * them to take the global cgroup_mutex mutex when exiting.
+ * This could impact scaling on very large systems.  Be reluctant to
+ * use notify_on_release cgroups where very high task exit scaling
+ * is required on large systems.
+ *
+ * the_top_cgroup_hack:
+ *
+ *    Set the exiting tasks cgroup to the root cgroup (top_cgroup).
+ *
+ *    We call cgroup_exit() while the task is still competent to
+ *    handle notify_on_release(), then leave the task attached to the
+ *    root cgroup in each hierarchy for the remainder of its exit.
+ *
+ *    To do this properly, we would increment the reference count on
+ *    top_cgroup, and near the very end of the kernel/exit.c do_exit()
+ *    code we would add a second cgroup function call, to drop that
+ *    reference.  This would just create an unnecessary hot spot on
+ *    the top_cgroup reference count, to no avail.
+ *
+ *    Normally, holding a reference to a cgroup without bumping its
+ *    count is unsafe.   The cgroup could go away, or someone could
+ *    attach us to a different cgroup, decrementing the count on
+ *    the first cgroup that we never incremented.  But in this case,
+ *    top_cgroup isn't going away, and either task has PF_EXITING set,
+ *    which wards off any attach_task() attempts, or task is a failed
+ *    fork, never visible to attach_task.
+ *
+ */
+void cgroup_exit(struct task_struct *tsk, int run_callbacks)
+{
+ int i;
+
+ if (run_callbacks && need_forkexit_callback) {
+  for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
```

```
+   struct cgroup_subsys *ss = subsys[i];
+   if (ss->exit)
+    ss->exit(ss, tsk);
+  }
+ }
+ /* Reassign the task to the init_css_set. */
+ task_lock(tsk);
+ put_css_set(&tsk->cgroups);
+ tsk->cgroups = init_task.cgroups;
+ task_unlock(tsk);
+}
diff -puN kernel/exit.c~task-cgroupsv11-add-fork-exit-hooks kernel/exit.c
--- a/kernel/exit.c~task-cgroupsv11-add-fork-exit-hooks
+++ a/kernel/exit.c
@@ -33,6 +33,7 @@
 #include <linux/delayacct.h>
 #include <linux/freezer.h>
 #include <linux/cpuset.h>
+#include <linux/cgroup.h>
 #include <linux/syscalls.h>
 #include <linux/signal.h>
 #include <linux/posix-timers.h>
@@ -981,6 +982,7 @@ fastcall NORET_TYPE void do_exit(long co
  check_stack_usage();
  exit_thread();
  cpuset_exit(tsk);
+ cgroup_exit(tsk, 1);
  exit_keys(tsk);

  if (group_dead && tsk->signal->leader)
diff -puN kernel/fork.c~task-cgroupsv11-add-fork-exit-hooks kernel/fork.c
--- a/kernel/fork.c~task-cgroupsv11-add-fork-exit-hooks
+++ a/kernel/fork.c
@@ -30,6 +30,7 @@
 #include <linux/capability.h>
 #include <linux/cpu.h>
 #include <linux/cpuset.h>
+#include <linux/cgroup.h>
 #include <linux/security.h>
 #include <linux/swap.h>
 #include <linux/syscalls.h>
@@ -967,6 +968,7 @@ static struct task_struct *copy_process(
 {
  int retval;
  struct task_struct *p = NULL;
+ int cgroup_callbacks_done = 0;

  if ((clone_flags & (CLONE_NEWNS|CLONE_FS)) == (CLONE_NEWNS|CLONE_FS))
```

```
   return ERR_PTR(-EINVAL);
@@ -1068,12 +1070,13 @@ static struct task_struct *copy_process(
  p->io_context = NULL;
  p->audit_context = NULL;
  cpuset_fork(p);
+ cgroup_fork(p);
 #ifdef CONFIG_NUMA
   p->mempolicy = mpol_copy(p->mempolicy);
   if (IS_ERR(p->mempolicy)) {
    retval = PTR_ERR(p->mempolicy);
    p->mempolicy = NULL;
-   goto bad_fork_cleanup_cpuset;
+   goto bad_fork_cleanup_cgroup;
   }
  mpol_fix_fork_child_flag(p);
 #endif
@@ -1184,6 +1187,12 @@ static struct task_struct *copy_process(
  /* Perform scheduler related setup. Assign this task to a CPU. */
  sched_fork(p, clone_flags);

+ /* Now that the task is set up, run cgroup callbacks if
+  * necessary. We need to run them before the task is visible
+  * on the tasklist. */
+ cgroup_fork_callbacks(p);
+ cgroup_callbacks_done = 1;
+
  /* Need tasklist lock for parent etc handling! */
  write_lock_irq(&tasklist_lock);

@@ -1306,9 +1315,10 @@ bad_fork_cleanup_security:
 bad_fork_cleanup_policy:
 #ifdef CONFIG_NUMA
  mpol_free(p->mempolicy);
-bad_fork_cleanup_cpuset:
+bad_fork_cleanup_cgroup:
 #endif
  cpuset_exit(p);
+ cgroup_exit(p, cgroup_callbacks_done);
  delayacct_tsk_free(p);
  if (p->binfmt)
   module_put(p->binfmt->module);
_

--
```

Subject: [PATCH 05/29] task containersv11 add container_clone interface
Posted by Paul Menage on Tue, 11 Sep 2007 19:52:44 GMT
View Forum Message <> Reply to Message

From: Paul Menage <menage@google.com>

Add support for cgroup_clone(), a way to create new cgroups intended to
be used for systems such as namespace unsharing. A new subsystem callback,
post_clone(), is added to allow subsystems to automatically configure cloned
cgroups.

Signed-off-by: Paul Menage <menage@google.com>
Cc: Serge E. Hallyn <serue@us.ibm.com>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Dave Hansen <haveblue@us.ibm.com>
Cc: Balbir Singh <balbir@in.ibm.com>
Cc: Paul Jackson <pj@sgi.com>
Cc: Kirill Korotaev <dev@openvz.org>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: Srivatsa Vaddagiri <vatsa@in.ibm.com>
Cc: Cedric Le Goater <clg@fr.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>

Documentation/cgroups.txt | 7 +
include/linux/cgroup.h | 3
kernel/cgroup.c | 135 ++
3 files changed, 145 insertions(+)

diff -puN Documentation/cgroups.txt~task-cgroupsv11-add-cgroup_clone-interface Documentation/cgroups.txt
--- a/Documentation/cgroups.txt~task-cgroupsv11-add-cgroup_clone-interface
+++ a/Documentation/cgroups.txt
@@ -504,6 +504,13 @@ include/linux/cgroup.h for details).
method can return an error code, the error code is currently not
always handled well.

+void post_clone(struct cgroup_subsys *ss, struct cgroup *cont)
+
+Called at the end of cgroup_clone() to do any paramater
+initialization which might be required before a task could attach. For
+example in cpusets, no task may attach before 'cpus' and 'mems' are set
+up.
+
void bind(struct cgroup_subsys *ss, struct cgroup *root)
LL=callback_mutex

diff -puN include/linux/cgroup.h~task-cgroupsv11-add-cgroup_clone-interface include/linux/cgroup.h

```
--- a/include/linux/cgroup.h~task-cgroupsv11-add-cgroup_clone-interface
+++ a/include/linux/cgroup.h
@@ -174,6 +174,7 @@ struct cgroup_subsys {
  void (*exit)(struct cgroup_subsys *ss, struct task_struct *task);
  int (*populate)(struct cgroup_subsys *ss,
    struct cgroup *cont);
+ void (*post_clone)(struct cgroup_subsys *ss, struct cgroup *cont);
  void (*bind)(struct cgroup_subsys *ss, struct cgroup *root);
  int subsys_id;
  int active;
@@ -213,6 +214,8 @@ static inline struct cgroup* task_con

 int cgroup_path(const struct cgroup *cont, char *buf, int buflen);

+int cgroup_clone(struct task_struct *tsk, struct cgroup_subsys *ss);
+
 #else /* !CONFIG_CGROUPS */

 static inline int cgroup_init_early(void) { return 0; }
diff -puN kernel/cgroup.c~task-cgroupsv11-add-cgroup_clone-interface kernel/cgroup.c
--- a/kernel/cgroup.c~task-cgroupsv11-add-cgroup_clone-interface
+++ a/kernel/cgroup.c
@@ -1684,3 +1684,138 @@ void cgroup_exit(struct task_struct *
  tsk->cgroups = init_task.cgroups;
  task_unlock(tsk);
 }
+
+/**
+ * cgroup_clone - duplicate the current cgroup in the hierarchy
+ * that the given subsystem is attached to, and move this task into
+ * the new child
+ */
+int cgroup_clone(struct task_struct *tsk, struct cgroup_subsys *subsys)
+{
+ struct dentry *dentry;
+ int ret = 0;
+ char nodename[MAX_CGROUP_TYPE_NAMELEN];
+ struct cgroup *parent, *child;
+ struct inode *inode;
+ struct css_set *cg;
+ struct cgroupfs_root *root;
+ struct cgroup_subsys *ss;
+
+ /* We shouldn't be called by an unregistered subsystem */
+ BUG_ON(!subsys->active);
+
+ /* First figure out what hierarchy and cgroup we're dealing
+  * with, and pin them so we can drop cgroup_mutex */
```

```
+ mutex_lock(&cgroup_mutex);
+ again:
+ root = subsys->root;
+ if (root == &rootnode) {
+  printk(KERN_INFO
+         "Not cloning cgroup for unused subsystem %s\n",
+         subsys->name);
+  mutex_unlock(&cgroup_mutex);
+  return 0;
+ }
+ cg = &tsk->cgroups;
+ parent = task_cgroup(tsk, subsys->subsys_id);
+
+ snprintf(nodename, MAX_CGROUP_TYPE_NAMELEN, "node_%d", tsk->pid);
+
+ /* Pin the hierarchy */
+ atomic_inc(&parent->root->sb->s_active);
+
+ mutex_unlock(&cgroup_mutex);
+
+ /* Now do the VFS work to create a cgroup */
+ inode = parent->dentry->d_inode;
+
+ /* Hold the parent directory mutex across this operation to
+  * stop anyone else deleting the new cgroup */
+ mutex_lock(&inode->i_mutex);
+ dentry = cgroup_get_dentry(parent->dentry, nodename);
+ if (IS_ERR(dentry)) {
+  printk(KERN_INFO
+         "Couldn't allocate dentry for %s: %ld\n", nodename,
+         PTR_ERR(dentry));
+  ret = PTR_ERR(dentry);
+  goto out_release;
+ }
+
+ /* Create the cgroup directory, which also creates the cgroup */
+ ret = vfs_mkdir(inode, dentry, S_IFDIR | 0755);
+ child = __d_cont(dentry);
+ dput(dentry);
+ if (ret) {
+  printk(KERN_INFO
+         "Failed to create cgroup %s: %d\n", nodename,
+         ret);
+  goto out_release;
+ }
+
+ if (!child) {
+  printk(KERN_INFO
```

```
+         "Couldn't find new cgroup %s\n", nodename);
+  ret = -ENOMEM;
+  goto out_release;
+ }
+
+ /* The cgroup now exists. Retake cgroup_mutex and check
+  * that we're still in the same state that we thought we
+  * were. */
+ mutex_lock(&cgroup_mutex);
+ if ((root != subsys->root) ||
+     (parent != task_cgroup(tsk, subsys->subsys_id))) {
+  /* Aargh, we raced ... */
+  mutex_unlock(&inode->i_mutex);
+
+  deactivate_super(parent->root->sb);
+  /* The cgroup is still accessible in the VFS, but
+   * we're not going to try to rmdir() it at this
+   * point. */
+  printk(KERN_INFO
+         "Race in cgroup_clone() - leaking cgroup %s\n",
+         nodename);
+  goto again;
+ }
+
+ /* do any required auto-setup */
+ for_each_subsys(root, ss) {
+  if (ss->post_clone)
+   ss->post_clone(ss, child);
+ }
+
+ /* All seems fine. Finish by moving the task into the new cgroup */
+ ret = attach_task(child, tsk);
+ mutex_unlock(&cgroup_mutex);
+
+ out_release:
+ mutex_unlock(&inode->i_mutex);
+ deactivate_super(parent->root->sb);
+ return ret;
+}
+
+/*
+ * See if "cont" is a descendant of the current task's cgroup in
+ * the appropriate hierarchy
+ *
+ * If we are sending in dummytop, then presumably we are creating
+ * the top cgroup in the subsystem.
+ *
+ * Called only by the ns (nsproxy) cgroup.
```

```
+ */
+int cgroup_is_descendant(const struct cgroup *cont)
+{
+ int ret;
+ struct cgroup *target;
+ int subsys_id;
+
+ if (cont == dummytop)
+  return 1;
+
+ get_first_subsys(cont, NULL, &subsys_id);
+ target = task_cgroup(current, subsys_id);
+ while (cont != target && cont!= cont->top_cgroup)
+  cont = cont->parent;
+ ret = (cont == target);
+ return ret;
+}
```

–

--

Subject: [PATCH 06/29] task containersv11 add procfs interface
Posted by Paul Menage on Tue, 11 Sep 2007 19:52:45 GMT
View Forum Message <> Reply to Message

From: Paul Menage <menage@google.com>

Add:

/proc/cgroups - general system info

/proc/*/cgroup - per-task cgroup membership info

Signed-off-by: Paul Menage <menage@google.com>
Cc: Serge E. Hallyn <serue@us.ibm.com>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Dave Hansen <haveblue@us.ibm.com>
Cc: Balbir Singh <balbir@in.ibm.com>
Cc: Paul Jackson <pj@sgi.com>
Cc: Kirill Korotaev <dev@openvz.org>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: Srivatsa Vaddagiri <vatsa@in.ibm.com>
Cc: Cedric Le Goater <clg@fr.ibm.com>

```
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

 fs/proc/base.c           |    7 +
 include/linux/cgroup.h |    2
 kernel/cgroup.c        |  132 ++++++++++++++++++++++++++++++++++++++++++++++++
 3 files changed, 141 insertions(+)

diff -puN fs/proc/base.c~task-cgroupsv11-add-procfs-interface fs/proc/base.c
--- a/fs/proc/base.c~task-cgroupsv11-add-procfs-interface
+++ a/fs/proc/base.c
@@ -67,6 +67,7 @@
 #include <linux/mount.h>
 #include <linux/security.h>
 #include <linux/ptrace.h>
+#include <linux/cgroup.h>
 #include <linux/cpuset.h>
 #include <linux/audit.h>
 #include <linux/poll.h>
@@ -2051,6 +2052,9 @@ static const struct pid_entry tgid_base_
 #ifdef CONFIG_CPUSETS
   REG("cpuset",     S_IRUGO, cpuset),
 #endif
+#ifdef CONFIG_CGROUPS
+ REG("cgroup",  S_IRUGO, cgroup),
+#endif
   INF("oom_score",  S_IRUGO, oom_score),
   REG("oom_adj",    S_IRUGO|S_IWUSR, oom_adjust),
 #ifdef CONFIG_AUDITSYSCALL
@@ -2340,6 +2344,9 @@ static const struct pid_entry tid_base_s
 #ifdef CONFIG_CPUSETS
   REG("cpuset",    S_IRUGO, cpuset),
 #endif
+#ifdef CONFIG_CGROUPS
+ REG("cgroup",  S_IRUGO, cgroup),
+#endif
   INF("oom_score", S_IRUGO, oom_score),
   REG("oom_adj",   S_IRUGO|S_IWUSR, oom_adjust),
 #ifdef CONFIG_AUDITSYSCALL
diff -puN include/linux/cgroup.h~task-cgroupsv11-add-procfs-interface include/linux/cgroup.h
--- a/include/linux/cgroup.h~task-cgroupsv11-add-procfs-interface
+++ a/include/linux/cgroup.h
@@ -29,6 +29,8 @@ extern void cgroup_fork(struct task_s
 extern void cgroup_fork_callbacks(struct task_struct *p);
 extern void cgroup_exit(struct task_struct *p, int run_callbacks);

+extern struct file_operations proc_cgroup_operations;
+
```

```
 /* Per-subsystem/per-cgroup state maintained by the system. */
 struct cgroup_subsys_state {
  /* The cgroup that this subsystem is attached to. Useful
diff -puN kernel/cgroup.c~task-cgroupsv11-add-procfs-interface kernel/cgroup.c
--- a/kernel/cgroup.c~task-cgroupsv11-add-procfs-interface
+++ a/kernel/cgroup.c
@@ -33,6 +33,7 @@
 #include <linux/mutex.h>
 #include <linux/mount.h>
 #include <linux/pagemap.h>
+#include <linux/proc_fs.h>
 #include <linux/rcupdate.h>
 #include <linux/sched.h>
 #include <linux/seq_file.h>
@@ -247,6 +248,7 @@ static int cgroup_mkdir(struct inode
 static int cgroup_rmdir(struct inode *unused_dir, struct dentry *dentry);
 static int cgroup_populate_dir(struct cgroup *cont);
 static struct inode_operations cgroup_dir_inode_operations;
+static struct file_operations proc_cgroupstats_operations;

 static struct inode *cgroup_new_inode(mode_t mode, struct super_block *sb)
 {
@@ -1576,6 +1578,7 @@ int __init cgroup_init(void)
 {
  int err;
  int i;
+ struct proc_dir_entry *entry;

  for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
   struct cgroup_subsys *ss = subsys[i];
@@ -1587,10 +1590,139 @@ int __init cgroup_init(void)
  if (err < 0)
   goto out;

+ entry = create_proc_entry("cgroups", 0, NULL);
+ if (entry)
+  entry->proc_fops = &proc_cgroupstats_operations;
+
 out:
  return err;
 }

+/*
+ * proc_cgroup_show()
+ *  - Print task's cgroup paths into seq_file, one line for each hierarchy
+ *  - Used for /proc/<pid>/cgroup.
+ *  - No need to task_lock(tsk) on this tsk->cgroup reference, as it
+ *    doesn't really matter if tsk->cgroup changes after we read it,
```

```
+ *    and we take cgroup_mutex, keeping attach_task() from changing it
+ *    anyway.  No need to check that tsk->cgroup != NULL, thanks to
+ *    the_top_cgroup_hack in cgroup_exit(), which sets an exiting tasks
+ *    cgroup to top_cgroup.
+ */
+
+/* TODO: Use a proper seq_file iterator */
+static int proc_cgroup_show(struct seq_file *m, void *v)
+{
+ struct pid *pid;
+ struct task_struct *tsk;
+ char *buf;
+ int retval;
+ struct cgroupfs_root *root;
+
+ retval = -ENOMEM;
+ buf = kmalloc(PAGE_SIZE, GFP_KERNEL);
+ if (!buf)
+  goto out;
+
+ retval = -ESRCH;
+ pid = m->private;
+ tsk = get_pid_task(pid, PIDTYPE_PID);
+ if (!tsk)
+  goto out_free;
+
+ retval = 0;
+
+ mutex_lock(&cgroup_mutex);
+
+ for_each_root(root) {
+  struct cgroup_subsys *ss;
+  struct cgroup *cont;
+  int subsys_id;
+  int count = 0;
+
+  /* Skip this hierarchy if it has no active subsystems */
+  if (!root->actual_subsys_bits)
+   continue;
+  for_each_subsys(root, ss)
+   seq_printf(m, "%s%s", count++ ? "," : "", ss->name);
+  seq_putc(m, ':');
+  get_first_subsys(&root->top_cgroup, NULL, &subsys_id);
+  cont = task_cgroup(tsk, subsys_id);
+  retval = cgroup_path(cont, buf, PAGE_SIZE);
+  if (retval < 0)
+   goto out_unlock;
+  seq_puts(m, buf);
```

```
+  seq_putc(m, '\n');
+ }
+
+out_unlock:
+ mutex_unlock(&cgroup_mutex);
+ put_task_struct(tsk);
+out_free:
+ kfree(buf);
+out:
+ return retval;
+}
+
+static int cgroup_open(struct inode *inode, struct file *file)
+{
+ struct pid *pid = PROC_I(inode)->pid;
+ return single_open(file, proc_cgroup_show, pid);
+}
+
+struct file_operations proc_cgroup_operations = {
+ .open  = cgroup_open,
+ .read  = seq_read,
+ .llseek  = seq_lseek,
+ .release = single_release,
+};
+
+/* Display information about each subsystem and each hierarchy */
+static int proc_cgroupstats_show(struct seq_file *m, void *v)
+{
+ int i;
+ struct cgroupfs_root *root;
+
+ mutex_lock(&cgroup_mutex);
+ seq_puts(m, "Hierarchies:\n");
+ for_each_root(root) {
+  struct cgroup_subsys *ss;
+  int first = 1;
+  seq_printf(m, "%p: bits=%lx cgroups=%d (", root,
+      root->subsys_bits, root->number_of_cgroups);
+  for_each_subsys(root, ss) {
+   seq_printf(m, "%s%s", first ? "" : ", ", ss->name);
+   first = false;
+  }
+  seq_putc(m, ')');
+  if (root->sb) {
+   seq_printf(m, " s_active=%d",
+       atomic_read(&root->sb->s_active));
+  }
+  seq_putc(m, '\n');
```

```
+ }
+ seq_puts(m, "Subsystems:\n");
+ for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
+  struct cgroup_subsys *ss = subsys[i];
+  seq_printf(m, "%d: name=%s hierarchy=%p\n",
+      i, ss->name, ss->root);
+ }
+ mutex_unlock(&cgroup_mutex);
+ return 0;
+}
+
+static int cgroupstats_open(struct inode *inode, struct file *file)
+{
+ return single_open(file, proc_cgroupstats_show, 0);
+}
+
+static struct file_operations proc_cgroupstats_operations = {
+ .open = cgroupstats_open,
+ .read = seq_read,
+ .llseek = seq_lseek,
+ .release = single_release,
+};
+
 /**
  * cgroup_fork - attach newly forked task to its parents cgroup.
  * @tsk: pointer to task_struct of forking parent process.
```

_

--

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 07/29] task containersv11 shared container subsystem group arrays

Posted by Paul Menage on Tue, 11 Sep 2007 19:52:46 GMT
View Forum Message <> Reply to Message

From: Paul Menage <menage@google.com>

Replace the struct css_set embedded in task_struct with a pointer; all tasks that have the same set of memberships across all hierarchies will share a css_set object, and will be linked via their css_sets field to the "tasks" list_head in the css_set.

Assuming that many tasks share the same cgroup assignments, this reduces

overall space usage and keeps the size of the task_struct down (three pointers
added to task_struct compared to a non-cgroups kernel, no matter how many
subsystems are registered).

Signed-off-by: Paul Menage <menage@google.com>
Cc: Serge E. Hallyn <serue@us.ibm.com>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Dave Hansen <haveblue@us.ibm.com>
Cc: Balbir Singh <balbir@in.ibm.com>
Cc: Paul Jackson <pj@sgi.com>
Cc: Kirill Korotaev <dev@openvz.org>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: Srivatsa Vaddagiri <vatsa@in.ibm.com>
Cc: Cedric Le Goater <clg@fr.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>

 Documentation/cgroups.txt | 14
 include/linux/cgroup.h | 89 ++++
 include/linux/sched.h | 33 -
 kernel/cgroup.c | 606 ++++++++++++++++++++++++++++++-----
 kernel/fork.c | 1
 5 files changed, 620 insertions(+), 123 deletions(-)

diff -puN Documentation/cgroups.txt~task-cgroupsv11-shared-cgroup-subsystem-group-arrays
Documentation/cgroups.txt
--- a/Documentation/cgroups.txt~task-cgroupsv11-shared-cgroup-subsystem-group-arrays
+++ a/Documentation/cgroups.txt
@@ -176,7 +176,9 @@ Control Groups extends the kernel as follows
    subsystem state is something that's expected to happen frequently
    and in performance-critical code, whereas operations that require a
    task's actual cgroup assignments (in particular, moving between
-   cgroups) are less common.
+   cgroups) are less common. A linked list runs through the cg_list
+   field of each task_struct using the css_set, anchored at
+   css_set->tasks.

  - A cgroup hierarchy filesystem can be mounted for browsing and
    manipulation from user space.
@@ -252,6 +254,16 @@ linear search to locate an appropriate e
 very efficient. A future version will use a hash table for better
 performance.

+To allow access from a cgroup to the css_sets (and hence tasks)
+that comprise it, a set of cg_cgroup_link objects form a lattice;
+each cg_cgroup_link is linked into a list of cg_cgroup_links for
+a single cgroup on its cont_link_list field, and a list of
+cg_cgroup_links for a single css_set on its cg_link_list.

```
+
+Thus the set of tasks in a cgroup can be listed by iterating over
+each css_set that references the cgroup, and sub-iterating over
+each css_set's task set.
+
 The use of a Linux virtual file system (vfs) to represent the
 cgroup hierarchy provides for a familiar permission and name space
 for cgroups, with a minimum of additional kernel code.
diff -puN include/linux/cgroup.h~task-cgroupsv11-shared-cgroup-subsystem-group-arrays
include/linux/cgroup.h
--- a/include/linux/cgroup.h~task-cgroupsv11-shared-cgroup-subsystem-group-arrays
+++ a/include/linux/cgroup.h
@@ -27,10 +27,19 @@ extern void cgroup_lock(void);
 extern void cgroup_unlock(void);
 extern void cgroup_fork(struct task_struct *p);
 extern void cgroup_fork_callbacks(struct task_struct *p);
+extern void cgroup_post_fork(struct task_struct *p);
 extern void cgroup_exit(struct task_struct *p, int run_callbacks);

 extern struct file_operations proc_cgroup_operations;

+/* Define the enumeration of all cgroup subsystems */
+#define SUBSYS(_x) _x ## _subsys_id,
+enum cgroup_subsys_id {
+#include <linux/cgroup_subsys.h>
+ CGROUP_SUBSYS_COUNT
+};
+#undef SUBSYS
+
 /* Per-subsystem/per-cgroup state maintained by the system. */
 struct cgroup_subsys_state {
  /* The cgroup that this subsystem is attached to. Useful
@@ -97,6 +106,52 @@ struct cgroup {

  struct cgroupfs_root *root;
  struct cgroup *top_cgroup;
+
+ /*
+  * List of cg_cgroup_links pointing at css_sets with
+  * tasks in this cgroup. Protected by css_set_lock
+  */
+ struct list_head css_sets;
+};
+
+/* A css_set is a structure holding pointers to a set of
+ * cgroup_subsys_state objects. This saves space in the task struct
+ * object and speeds up fork()/exit(), since a single inc/dec and a
+ * list_add()/del() can bump the reference count on the entire
```

```
+ * cgroup set for a task.
+ */
+
+struct css_set {
+
+ /* Reference count */
+ struct kref ref;
+
+ /*
+  * List running through all cgroup groups. Protected by
+  * css_set_lock
+  */
+ struct list_head list;
+
+ /*
+  * List running through all tasks using this cgroup
+  * group. Protected by css_set_lock
+  */
+ struct list_head tasks;
+
+ /*
+  * List of cg_cgroup_link objects on link chains from
+  * cgroups referenced from this css_set. Protected by
+  * css_set_lock
+  */
+ struct list_head cg_links;
+
+ /*
+  * Set of subsystem states, one for each subsystem. This array
+  * is immutable after creation apart from the init_css_set
+  * during subsystem registration (at boot time).
+  */
+ struct cgroup_subsys_state *subsys[CGROUP_SUBSYS_COUNT];
+
 };

 /* struct cftype:
@@ -149,15 +204,7 @@ int cgroup_is_removed(const struct co

 int cgroup_path(const struct cgroup *cont, char *buf, int buflen);

-int __cgroup_task_count(const struct cgroup *cont);
-static inline int cgroup_task_count(const struct cgroup *cont)
-{
- int task_count;
- rcu_read_lock();
- task_count = __cgroup_task_count(cont);
- rcu_read_unlock();
```

```
- return task_count;
-}
+int cgroup_task_count(const struct cgroup *cont);

 /* Return true if the cgroup is a descendant of the current cgroup */
 int cgroup_is_descendant(const struct cgroup *cont);
@@ -205,7 +252,7 @@ static inline struct cgroup_subsys_st
 static inline struct cgroup_subsys_state *task_subsys_state(
  struct task_struct *task, int subsys_id)
 {
- return rcu_dereference(task->cgroups.subsys[subsys_id]);
+ return rcu_dereference(task->cgroups->subsys[subsys_id]);
 }

 static inline struct cgroup* task_cgroup(struct task_struct *task,
@@ -218,6 +265,27 @@ int cgroup_path(const struct containe

 int cgroup_clone(struct task_struct *tsk, struct cgroup_subsys *ss);

+/* A cgroup_iter should be treated as an opaque object */
+struct cgroup_iter {
+ struct list_head *cg_link;
+ struct list_head *task;
+};
+
+/* To iterate across the tasks in a cgroup:
+ *
+ * 1) call cgroup_iter_start to intialize an iterator
+ *
+ * 2) call cgroup_iter_next() to retrieve member tasks until it
+ *    returns NULL or until you want to end the iteration
+ *
+ * 3) call cgroup_iter_end() to destroy the iterator.
+ */
+void cgroup_iter_start(struct cgroup *cont, struct cgroup_iter *it);
+struct task_struct *cgroup_iter_next(struct cgroup *cont,
+     struct cgroup_iter *it);
+void cgroup_iter_end(struct cgroup *cont, struct cgroup_iter *it);
+
+
 #else /* !CONFIG_CGROUPS */

 static inline int cgroup_init_early(void) { return 0; }
@@ -225,6 +293,7 @@ static inline int cgroup_init(void) {
 static inline void cgroup_init_smp(void) {}
 static inline void cgroup_fork(struct task_struct *p) {}
 static inline void cgroup_fork_callbacks(struct task_struct *p) {}
+static inline void cgroup_post_fork(struct task_struct *p) {}
```

```
 static inline void cgroup_exit(struct task_struct *p, int callbacks) {}

 static inline void cgroup_lock(void) {}
diff -puN include/linux/sched.h~task-cgroupsv11-shared-cgroup-subsystem-group-arrays
include/linux/sched.h
--- a/include/linux/sched.h~task-cgroupsv11-shared-cgroup-subsystem-group-arrays
+++ a/include/linux/sched.h
@@ -861,34 +861,6 @@ struct sched_entity {
 #endif
 };

-#ifdef CONFIG_CGROUPS
-
-#define SUBSYS(_x) _x ## _subsys_id,
-enum cgroup_subsys_id {
-#include <linux/cgroup_subsys.h>
- CGROUP_SUBSYS_COUNT
-};
-#undef SUBSYS
-
-/* A css_set is a structure holding pointers to a set of
- * cgroup_subsys_state objects.
- */
-
-struct css_set {
-
- /* Set of subsystem states, one for each subsystem. NULL for
-  * subsystems that aren't part of this hierarchy. These
-  * pointers reduce the number of dereferences required to get
-  * from a task to its state for a given cgroup, but result
-  * in increased space usage if tasks are in wildly different
-  * groupings across different hierarchies. This array is
-  * immutable after creation */
- struct cgroup_subsys_state *subsys[CGROUP_SUBSYS_COUNT];
-
-};
-
-#endif /* CONFIG_CGROUPS */
-
 struct task_struct {
  volatile long state; /* -1 unrunnable, 0 runnable, >0 stopped */
  void *stack;
@@ -1125,7 +1097,10 @@ struct task_struct {
  int cpuset_mem_spread_rotor;
 #endif
 #ifdef CONFIG_CGROUPS
- struct css_set cgroups;
+ /* Control Group info protected by css_set_lock */
```

```
+ struct css_set *cgroups;
+ /* cg_list protected by css_set_lock and tsk->alloc_lock */
+ struct list_head cg_list;
 #endif
 #ifdef CONFIG_FUTEX
  struct robust_list_head __user *robust_list;
diff -puN kernel/cgroup.c~task-cgroupsv11-shared-cgroup-subsystem-group-arrays
kernel/cgroup.c
--- a/kernel/cgroup.c~task-cgroupsv11-shared-cgroup-subsystem-group-arrays
+++ a/kernel/cgroup.c
@@ -95,6 +95,7 @@ static struct cgroupfs_root rootnode;
 /* The list of hierarchy roots */

 static LIST_HEAD(roots);
+static int root_count;

 /* dummytop is a shorthand for the dummy hierarchy's top cgroup */
 #define dummytop (&rootnode.top_cgroup)
@@ -133,12 +134,49 @@ list_for_each_entry(_ss, &_root->subsys_
 #define for_each_root(_root) \
 list_for_each_entry(_root, &roots, root_list)

-/* Each task_struct has an embedded css_set, so the get/put
- * operation simply takes a reference count on all the cgroups
- * referenced by subsystems in this css_set. This can end up
- * multiple-counting some cgroups, but that's OK - the ref-count is
- * just a busy/not-busy indicator; ensuring that we only count each
- * cgroup once would require taking a global lock to ensure that no
+/* Link structure for associating css_set objects with cgroups */
+struct cg_cgroup_link {
+ /*
+  * List running through cg_cgroup_links associated with a
+  * cgroup, anchored on cgroup->css_sets
+  */
+ struct list_head cont_link_list;
+ /*
+  * List running through cg_cgroup_links pointing at a
+  * single css_set object, anchored on css_set->cg_links
+  */
+ struct list_head cg_link_list;
+ struct css_set *cg;
+};
+
+/* The default css_set - used by init and its children prior to any
+ * hierarchies being mounted. It contains a pointer to the root state
+ * for each subsystem. Also used to anchor the list of css_sets. Not
+ * reference-counted, to improve performance when child cgroups
+ * haven't been created.
```

```
+ */
+
+static struct css_set init_css_set;
+static struct cg_cgroup_link init_css_set_link;
+
+/* css_set_lock protects the list of css_set objects, and the
+ * chain of tasks off each css_set.  Nests outside task->alloc_lock
+ * due to cgroup_iter_start() */
+static DEFINE_RWLOCK(css_set_lock);
+static int css_set_count;
+
+/* We don't maintain the lists running through each css_set to its
+ * task until after the first call to cgroup_iter_start(). This
+ * reduces the fork()/exit() overhead for people who have cgroups
+ * compiled into their kernel but not actually in use */
+static int use_task_css_set_links;
+
+/* When we create or destroy a css_set, the operation simply
+ * takes/releases a reference count on all the cgroups referenced
+ * by subsystems in this css_set. This can end up multiple-counting
+ * some cgroups, but that's OK - the ref-count is just a
+ * busy/not-busy indicator; ensuring that we only count each cgroup
+ * once would require taking a global lock to ensure that no
  * subsystems moved between hierarchies while we were doing so.
  *
  * Possible TODO: decide at boot time based on the number of
@@ -146,18 +184,230 @@ list_for_each_entry(_root, &roots, root_
  * it's better for performance to ref-count every subsystem, or to
  * take a global lock and only add one ref count to each hierarchy.
  */
-static void get_css_set(struct css_set *cg)
+
+/*
+ * unlink a css_set from the list and free it
+ */
+static void release_css_set(struct kref *k)
 {
+ struct css_set *cg = container_of(k, struct css_set, ref);
  int i;
+
+ write_lock(&css_set_lock);
+ list_del(&cg->list);
+ css_set_count--;
+ while (!list_empty(&cg->cg_links)) {
+  struct cg_cgroup_link *link;
+  link = list_entry(cg->cg_links.next,
+       struct cg_cgroup_link, cg_link_list);
+  list_del(&link->cg_link_list);
```

```
+  list_del(&link->cont_link_list);
+  kfree(link);
+ }
+ write_unlock(&css_set_lock);
  for (i = 0; i < CGROUP_SUBSYS_COUNT; i++)
-  atomic_inc(&cg->subsys[i]->cgroup->count);
+  atomic_dec(&cg->subsys[i]->cgroup->count);
+ kfree(cg);
+}
+
+/*
+ * refcounted get/put for css_set objects
+ */
+static inline void get_css_set(struct css_set *cg)
+{
+ kref_get(&cg->ref);
 }

-static void put_css_set(struct css_set *cg)
+static inline void put_css_set(struct css_set *cg)
+{
+ kref_put(&cg->ref, release_css_set);
+}
+
+/*
+ * find_existing_css_set() is a helper for
+ * find_css_set(), and checks to see whether an existing
+ * css_set is suitable. This currently walks a linked-list for
+ * simplicity; a later patch will use a hash table for better
+ * performance
+ *
+ * oldcg: the cgroup group that we're using before the cgroup
+ * transition
+ *
+ * cont: the cgroup that we're moving into
+ *
+ * template: location in which to build the desired set of subsystem
+ * state objects for the new cgroup group
+ */
+
+static struct css_set *find_existing_css_set(
+ struct css_set *oldcg,
+ struct cgroup *cont,
+ struct cgroup_subsys_state *template[])
 {
  int i;
- for (i = 0; i < CGROUP_SUBSYS_COUNT; i++)
-  atomic_dec(&cg->subsys[i]->cgroup->count);
```

```
+ struct cgroupfs_root *root = cont->root;
+ struct list_head *l = &init_css_set.list;
+
+ /* Built the set of subsystem state objects that we want to
+  * see in the new css_set */
+ for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
+  if (root->subsys_bits & (1ull << i)) {
+   /* Subsystem is in this hierarchy. So we want
+    * the subsystem state from the new
+    * cgroup */
+   template[i] = cont->subsys[i];
+  } else {
+   /* Subsystem is not in this hierarchy, so we
+    * don't want to change the subsystem state */
+   template[i] = oldcg->subsys[i];
+  }
+ }
+
+ /* Look through existing cgroup groups to find one to reuse */
+ do {
+  struct css_set *cg =
+   list_entry(l, struct css_set, list);
+
+  if (!memcmp(template, cg->subsys, sizeof(cg->subsys))) {
+   /* All subsystems matched */
+   return cg;
+  }
+  /* Try the next cgroup group */
+  l = l->next;
+ } while (l != &init_css_set.list);
+
+ /* No existing cgroup group matched */
+ return NULL;
+}
+
+/*
+ * allocate_cg_links() allocates "count" cg_cgroup_link structures
+ * and chains them on tmp through their cont_link_list fields. Returns 0 on
+ * success or a negative error
+ */
+
+static int allocate_cg_links(int count, struct list_head *tmp)
+{
+ struct cg_cgroup_link *link;
+ int i;
+ INIT_LIST_HEAD(tmp);
+ for (i = 0; i < count; i++) {
+  link = kmalloc(sizeof(*link), GFP_KERNEL);
```

```
+  if (!link) {
+   while (!list_empty(tmp)) {
+    link = list_entry(tmp->next,
+        struct cg_cgroup_link,
+        cont_link_list);
+    list_del(&link->cont_link_list);
+    kfree(link);
+   }
+   return -ENOMEM;
+  }
+  list_add(&link->cont_link_list, tmp);
+ }
+ return 0;
+}
+
+static void free_cg_links(struct list_head *tmp)
+{
+ while (!list_empty(tmp)) {
+  struct cg_cgroup_link *link;
+  link = list_entry(tmp->next,
+      struct cg_cgroup_link,
+      cont_link_list);
+  list_del(&link->cont_link_list);
+  kfree(link);
+ }
+}
+
+/*
+ * find_css_set() takes an existing cgroup group and a
+ * cgroup object, and returns a css_set object that's
+ * equivalent to the old group, but with the given cgroup
+ * substituted into the appropriate hierarchy. Must be called with
+ * cgroup_mutex held
+ */
+
+static struct css_set *find_css_set(
+ struct css_set *oldcg, struct cgroup *cont)
+{
+ struct css_set *res;
+ struct cgroup_subsys_state *template[CGROUP_SUBSYS_COUNT];
+ int i;
+
+ struct list_head tmp_cg_links;
+ struct cg_cgroup_link *link;
+
+ /* First see if we already have a cgroup group that matches
+  * the desired set */
+ write_lock(&css_set_lock);
```

```
+ res = find_existing_css_set(oldcg, cont, template);
+ if (res)
+  get_css_set(res);
+ write_unlock(&css_set_lock);
+
+ if (res)
+  return res;
+
+ res = kmalloc(sizeof(*res), GFP_KERNEL);
+ if (!res)
+  return NULL;
+
+ /* Allocate all the cg_cgroup_link objects that we'll need */
+ if (allocate_cg_links(root_count, &tmp_cg_links) < 0) {
+  kfree(res);
+  return NULL;
+ }
+
+ kref_init(&res->ref);
+ INIT_LIST_HEAD(&res->cg_links);
+ INIT_LIST_HEAD(&res->tasks);
+
+ /* Copy the set of subsystem state objects generated in
+  * find_existing_css_set() */
+ memcpy(res->subsys, template, sizeof(res->subsys));
+
+ write_lock(&css_set_lock);
+ /* Add reference counts and links from the new css_set. */
+ for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
+  struct cgroup *cont = res->subsys[i]->cgroup;
+  struct cgroup_subsys *ss = subsys[i];
+  atomic_inc(&cont->count);
+  /*
+   * We want to add a link once per cgroup, so we
+   * only do it for the first subsystem in each
+   * hierarchy
+   */
+  if (ss->root->subsys_list.next == &ss->sibling) {
+   BUG_ON(list_empty(&tmp_cg_links));
+   link = list_entry(tmp_cg_links.next,
+       struct cg_cgroup_link,
+       cont_link_list);
+   list_del(&link->cont_link_list);
+   list_add(&link->cont_link_list, &cont->css_sets);
+   link->cg = res;
+   list_add(&link->cg_link_list, &res->cg_links);
+  }
+ }
```

```
+ if (list_empty(&rootnode.subsys_list)) {
+  link = list_entry(tmp_cg_links.next,
+       struct cg_cgroup_link,
+       cont_link_list);
+  list_del(&link->cont_link_list);
+  list_add(&link->cont_link_list, &dummytop->css_sets);
+  link->cg = res;
+  list_add(&link->cg_link_list, &res->cg_links);
+ }
+
+ BUG_ON(!list_empty(&tmp_cg_links));
+
+ /* Link this cgroup group into the list */
+ list_add(&res->list, &init_css_set.list);
+ css_set_count++;
+ INIT_LIST_HEAD(&res->tasks);
+ write_unlock(&css_set_lock);
+
+ return res;
 }

 /*
@@ -516,6 +766,7 @@ static void init_cgroup_root(struct c
  cont->top_cgroup = cont;
  INIT_LIST_HEAD(&cont->sibling);
  INIT_LIST_HEAD(&cont->children);
+ INIT_LIST_HEAD(&cont->css_sets);
 }

 static int cgroup_test_super(struct super_block *sb, void *data)
@@ -585,6 +836,8 @@ static int cgroup_get_sb(struct file_
  int ret = 0;
  struct super_block *sb;
  struct cgroupfs_root *root;
+ struct list_head tmp_cg_links, *l;
+ INIT_LIST_HEAD(&tmp_cg_links);

  /* First find the desired set of subsystems */
  ret = parse_cgroupfs_options(data, &opts);
@@ -623,6 +876,19 @@ static int cgroup_get_sb(struct file_

   mutex_lock(&cgroup_mutex);

+  /*
+   * We're accessing css_set_count without locking
+   * css_set_lock here, but that's OK - it can only be
+   * increased by someone holding cgroup_lock, and
+   * that's us. The worst that can happen is that we
```

```
+   * have some link structures left over
+   */
+  ret = allocate_cg_links(css_set_count, &tmp_cg_links);
+  if (ret) {
+   mutex_unlock(&cgroup_mutex);
+   goto drop_new_super;
+  }
+
   ret = rebind_subsystems(root, root->subsys_bits);
   if (ret == -EBUSY) {
    mutex_unlock(&cgroup_mutex);
@@ -633,10 +899,34 @@ static int cgroup_get_sb(struct file_
   BUG_ON(ret);

   list_add(&root->root_list, &roots);
+  root_count++;

   sb->s_root->d_fsdata = &root->top_cgroup;
   root->top_cgroup.dentry = sb->s_root;

+  /* Link the top cgroup in this hierarchy into all
+   * the css_set objects */
+  write_lock(&css_set_lock);
+  l = &init_css_set.list;
+  do {
+   struct css_set *cg;
+   struct cg_cgroup_link *link;
+   cg = list_entry(l, struct css_set, list);
+   BUG_ON(list_empty(&tmp_cg_links));
+   link = list_entry(tmp_cg_links.next,
+       struct cg_cgroup_link,
+       cont_link_list);
+   list_del(&link->cont_link_list);
+   link->cg = cg;
+   list_add(&link->cont_link_list,
+     &root->top_cgroup.css_sets);
+   list_add(&link->cg_link_list, &cg->cg_links);
+   l = l->next;
+  } while (l != &init_css_set.list);
+  write_unlock(&css_set_lock);
+
+  free_cg_links(&tmp_cg_links);
+
   BUG_ON(!list_empty(&cont->sibling));
   BUG_ON(!list_empty(&cont->children));
   BUG_ON(root->number_of_cgroups != 1);
@@ -659,6 +949,7 @@ static int cgroup_get_sb(struct file_
  drop_new_super:
```

```
  up_write(&sb->s_umount);
  deactivate_super(sb);
+ free_cg_links(&tmp_cg_links);
  return ret;
 }

@@ -680,8 +971,25 @@ static void cgroup_kill_sb(struct sup
  /* Shouldn't be able to fail ... */
  BUG_ON(ret);

- if (!list_empty(&root->root_list))
+ /*
+  * Release all the links from css_sets to this hierarchy's
+  * root cgroup
+  */
+ write_lock(&css_set_lock);
+ while (!list_empty(&cont->css_sets)) {
+  struct cg_cgroup_link *link;
+  link = list_entry(cont->css_sets.next,
+       struct cg_cgroup_link, cont_link_list);
+  list_del(&link->cg_link_list);
+  list_del(&link->cont_link_list);
+  kfree(link);
+ }
+ write_unlock(&css_set_lock);
+
+ if (!list_empty(&root->root_list)) {
   list_del(&root->root_list);
+  root_count--;
+ }
  mutex_unlock(&cgroup_mutex);

  kfree(root);
@@ -774,9 +1082,9 @@ static int attach_task(struct cgroup
  int retval = 0;
  struct cgroup_subsys *ss;
  struct cgroup *oldcont;
- struct css_set *cg = &tsk->cgroups;
+ struct css_set *cg = tsk->cgroups;
+ struct css_set *newcg;
  struct cgroupfs_root *root = cont->root;
- int i;
  int subsys_id;

  get_first_subsys(cont, NULL, &subsys_id);
@@ -795,26 +1103,32 @@ static int attach_task(struct cgroup
   }
  }
```

```
+ /*
+  * Locate or allocate a new css_set for this task,
+  * based on its final set of cgroups
+  */
+ newcg = find_css_set(cg, cont);
+ if (!newcg) {
+  return -ENOMEM;
+ }
+
  task_lock(tsk);
  if (tsk->flags & PF_EXITING) {
   task_unlock(tsk);
+  put_css_set(newcg);
   return -ESRCH;
  }
- /* Update the css_set pointers for the subsystems in this
-  * hierarchy */
- for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
-  if (root->subsys_bits & (1ull << i)) {
-   /* Subsystem is in this hierarchy. So we want
-    * the subsystem state from the new
-    * cgroup. Transfer the refcount from the
-    * old to the new */
-   atomic_inc(&cont->count);
-   atomic_dec(&cg->subsys[i]->cgroup->count);
-   rcu_assign_pointer(cg->subsys[i], cont->subsys[i]);
-  }
- }
+ rcu_assign_pointer(tsk->cgroups, newcg);
  task_unlock(tsk);

+ /* Update the css_set linked lists if we're using them */
+ write_lock(&css_set_lock);
+ if (!list_empty(&tsk->cg_list)) {
+  list_del(&tsk->cg_list);
+  list_add(&tsk->cg_list, &newcg->tasks);
+ }
+ write_unlock(&css_set_lock);
+
  for_each_subsys(root, ss) {
   if (ss->attach) {
    ss->attach(ss, cont, oldcont, tsk);
@@ -822,6 +1136,7 @@ static int attach_task(struct cgroup
  }

  synchronize_rcu();
+ put_css_set(cg);
```

```
  return 0;
 }

@@ -1123,28 +1438,102 @@ int cgroup_add_files(struct cgroup
  return 0;
 }

-/* Count the number of tasks in a cgroup. Could be made more
- * time-efficient but less space-efficient with more linked lists
- * running through each cgroup and the css_set structures that
- * referenced it. Must be called with tasklist_lock held for read or
- * write or in an rcu critical section.
- */
-int __cgroup_task_count(const struct cgroup *cont)
+/* Count the number of tasks in a cgroup. */
+
+int cgroup_task_count(const struct cgroup *cont)
 {
  int count = 0;
- struct task_struct *g, *p;
- struct cgroup_subsys_state *css;
- int subsys_id;
+ struct list_head *l;

- get_first_subsys(cont, &css, &subsys_id);
- do_each_thread(g, p) {
-  if (task_subsys_state(p, subsys_id) == css)
-   count ++;
- } while_each_thread(g, p);
+ read_lock(&css_set_lock);
+ l = cont->css_sets.next;
+ while (l != &cont->css_sets) {
+  struct cg_cgroup_link *link =
+   list_entry(l, struct cg_cgroup_link, cont_link_list);
+  count += atomic_read(&link->cg->ref.refcount);
+  l = l->next;
+ }
+ read_unlock(&css_set_lock);
  return count;
 }

 /*
+ * Advance a list_head iterator.  The iterator should be positioned at
+ * the start of a css_set
+ */
+static void cgroup_advance_iter(struct cgroup *cont,
+       struct cgroup_iter *it)
+{
```

```
+ struct list_head *l = it->cg_link;
+ struct cg_cgroup_link *link;
+ struct css_set *cg;
+
+ /* Advance to the next non-empty css_set */
+ do {
+  l = l->next;
+  if (l == &cont->css_sets) {
+   it->cg_link = NULL;
+   return;
+  }
+  link = list_entry(l, struct cg_cgroup_link, cont_link_list);
+  cg = link->cg;
+ } while (list_empty(&cg->tasks));
+ it->cg_link = l;
+ it->task = cg->tasks.next;
+}
+
+void cgroup_iter_start(struct cgroup *cont, struct cgroup_iter *it)
+{
+ /*
+  * The first time anyone tries to iterate across a cgroup,
+  * we need to enable the list linking each css_set to its
+  * tasks, and fix up all existing tasks.
+  */
+ if (!use_task_css_set_links) {
+  struct task_struct *p, *g;
+  write_lock(&css_set_lock);
+  use_task_css_set_links = 1;
+   do_each_thread(g, p) {
+   task_lock(p);
+   if (list_empty(&p->cg_list))
+    list_add(&p->cg_list, &p->cgroups->tasks);
+   task_unlock(p);
+   } while_each_thread(g, p);
+  write_unlock(&css_set_lock);
+ }
+ read_lock(&css_set_lock);
+ it->cg_link = &cont->css_sets;
+ cgroup_advance_iter(cont, it);
+}
+
+struct task_struct *cgroup_iter_next(struct cgroup *cont,
+     struct cgroup_iter *it)
+{
+ struct task_struct *res;
+ struct list_head *l = it->task;
+
```

```
+ /* If the iterator cg is NULL, we have no tasks */
+ if (!it->cg_link)
+  return NULL;
+ res = list_entry(l, struct task_struct, cg_list);
+ /* Advance iterator to find next entry */
+ l = l->next;
+ if (l == &res->cgroups->tasks) {
+  /* We reached the end of this task list - move on to
+   * the next cg_cgroup_link */
+  cgroup_advance_iter(cont, it);
+ } else {
+  it->task = l;
+ }
+ return res;
+}
+
+void cgroup_iter_end(struct cgroup *cont, struct cgroup_iter *it)
+{
+ read_unlock(&css_set_lock);
+}
+
+/*
  * Stuff for reading the 'tasks' file.
  *
  * Reading this file can return large amounts of data if a cgroup has
@@ -1173,22 +1562,15 @@ struct ctr_struct {
 static int pid_array_load(pid_t *pidarray, int npids, struct cgroup *cont)
 {
  int n = 0;
- struct task_struct *g, *p;
- struct cgroup_subsys_state *css;
- int subsys_id;
-
- get_first_subsys(cont, &css, &subsys_id);
- rcu_read_lock();
- do_each_thread(g, p) {
-  if (task_subsys_state(p, subsys_id) == css) {
-   pidarray[n++] = pid_nr(task_pid(p));
-   if (unlikely(n == npids))
-    goto array_full;
-  }
- } while_each_thread(g, p);
-
-array_full:
- rcu_read_unlock();
+ struct cgroup_iter it;
+ struct task_struct *tsk;
+ cgroup_iter_start(cont, &it);
```

```
+ while ((tsk = cgroup_iter_next(cont, &it))) {
+  if (unlikely(n == npids))
+   break;
+  pidarray[n++] = pid_nr(task_pid(tsk));
+ }
+ cgroup_iter_end(cont, &it);
  return n;
 }

@@ -1373,6 +1755,7 @@ static long cgroup_create(struct cont
  cont->flags = 0;
  INIT_LIST_HEAD(&cont->sibling);
  INIT_LIST_HEAD(&cont->children);
+ INIT_LIST_HEAD(&cont->css_sets);

  cont->parent = parent;
  cont->root = parent->root;
@@ -1504,8 +1887,8 @@ static int cgroup_rmdir(struct inode

 static void cgroup_init_subsys(struct cgroup_subsys *ss)
 {
- struct task_struct *g, *p;
  struct cgroup_subsys_state *css;
+ struct list_head *l;
  printk(KERN_ERR "Initializing cgroup subsys %s\n", ss->name);

  /* Create the top cgroup state for this subsystem */
@@ -1515,26 +1898,32 @@ static void cgroup_init_subsys(struct
  BUG_ON(IS_ERR(css));
  init_cgroup_css(css, ss, dummytop);

- /* Update all tasks to contain a subsys pointer to this state
-  * - since the subsystem is newly registered, all tasks are in
-  * the subsystem's top cgroup. */
+ /* Update all cgroup groups to contain a subsys
+  * pointer to this state - since the subsystem is
+  * newly registered, all tasks and hence all cgroup
+  * groups are in the subsystem's top cgroup. */
+ write_lock(&css_set_lock);
+ l = &init_css_set.list;
+ do {
+  struct css_set *cg =
+   list_entry(l, struct css_set, list);
+  cg->subsys[ss->subsys_id] = dummytop->subsys[ss->subsys_id];
+  l = l->next;
+ } while (l != &init_css_set.list);
+ write_unlock(&css_set_lock);
```

```
   /* If this subsystem requested that it be notified with fork
    * events, we should send it one now for every process in the
    * system */
+  if (ss->fork) {
+   struct task_struct *g, *p;

- read_lock(&tasklist_lock);
- init_task.cgroups.subsys[ss->subsys_id] = css;
- if (ss->fork)
-  ss->fork(ss, &init_task);
-
- do_each_thread(g, p) {
-  printk(KERN_INFO "Setting task %p css to %p (%d)\n", css, p, p->pid);
-  p->cgroups.subsys[ss->subsys_id] = css;
-  if (ss->fork)
-   ss->fork(ss, p);
- } while_each_thread(g, p);
- read_unlock(&tasklist_lock);
+   read_lock(&tasklist_lock);
+   do_each_thread(g, p) {
+    ss->fork(ss, p);
+   } while_each_thread(g, p);
+   read_unlock(&tasklist_lock);
+  }

  need_forkexit_callback |= ss->fork || ss->exit;

@@ -1548,8 +1937,22 @@ static void cgroup_init_subsys(struct
 int __init cgroup_init_early(void)
 {
  int i;
+ kref_init(&init_css_set.ref);
+ kref_get(&init_css_set.ref);
+ INIT_LIST_HEAD(&init_css_set.list);
+ INIT_LIST_HEAD(&init_css_set.cg_links);
+ INIT_LIST_HEAD(&init_css_set.tasks);
+ css_set_count = 1;
  init_cgroup_root(&rootnode);
  list_add(&rootnode.root_list, &roots);
+ root_count = 1;
+ init_task.cgroups = &init_css_set;
+
+ init_css_set_link.cg = &init_css_set;
+ list_add(&init_css_set_link.cont_link_list,
+   &rootnode.top_cgroup.css_sets);
+ list_add(&init_css_set_link.cg_link_list,
+   &init_css_set.cg_links);
```

```
  for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
   struct cgroup_subsys *ss = subsys[i];
@@ -1707,6 +2110,7 @@ static int proc_cgroupstats_show(stru
   seq_printf(m, "%d: name=%s hierarchy=%p\n",
       i, ss->name, ss->root);
  }
+ seq_printf(m, "Control Group groups: %d\n", css_set_count);
  mutex_unlock(&cgroup_mutex);
  return 0;
 }
@@ -1733,18 +2137,19 @@ static struct file_operations proc_conta
  * fork.c by dup_task_struct().  However, we ignore that copy, since
  * it was not made under the protection of RCU or cgroup_mutex, so
  * might no longer be a valid cgroup pointer.  attach_task() might
- * have already changed current->cgroup, allowing the previously
- * referenced cgroup to be removed and freed.
+ * have already changed current->cgroups, allowing the previously
+ * referenced cgroup group to be removed and freed.
  *
  * At the point that cgroup_fork() is called, 'current' is the parent
  * task, and the passed argument 'child' points to the child task.
  */
 void cgroup_fork(struct task_struct *child)
 {
- rcu_read_lock();
- child->cgroups = rcu_dereference(current->cgroups);
- get_css_set(&child->cgroups);
- rcu_read_unlock();
+ task_lock(current);
+ child->cgroups = current->cgroups;
+ get_css_set(child->cgroups);
+ task_unlock(current);
+ INIT_LIST_HEAD(&child->cg_list);
 }

 /**
@@ -1765,6 +2170,21 @@ void cgroup_fork_callbacks(struct tas
 }

 /**
+ * cgroup_post_fork - called on a new task after adding it to the
+ * task list. Adds the task to the list running through its css_set
+ * if necessary. Has to be after the task is visible on the task list
+ * in case we race with the first call to cgroup_iter_start() - to
+ * guarantee that the new task ends up on its list. */
+void cgroup_post_fork(struct task_struct *child)
+{
+ if (use_task_css_set_links) {
```

```
+  write_lock(&css_set_lock);
+  if (list_empty(&child->cg_list))
+   list_add(&child->cg_list, &child->cgroups->tasks);
+  write_unlock(&css_set_lock);
+ }
+}
+/**
  * cgroup_exit - detach cgroup from exiting task
  * @tsk: pointer to task_struct of exiting process
  *
@@ -1802,6 +2222,7 @@ void cgroup_fork_callbacks(struct tas
 void cgroup_exit(struct task_struct *tsk, int run_callbacks)
 {
  int i;
+ struct css_set *cg;

  if (run_callbacks && need_forkexit_callback) {
   for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
@@ -1810,11 +2231,26 @@ void cgroup_exit(struct task_struct *
     ss->exit(ss, tsk);
   }
  }
+
+ /*
+  * Unlink from the css_set task list if necessary.
+  * Optimistically check cg_list before taking
+  * css_set_lock
+  */
+ if (!list_empty(&tsk->cg_list)) {
+  write_lock(&css_set_lock);
+  if (!list_empty(&tsk->cg_list))
+   list_del(&tsk->cg_list);
+  write_unlock(&css_set_lock);
+ }
+
  /* Reassign the task to the init_css_set. */
  task_lock(tsk);
- put_css_set(&tsk->cgroups);
- tsk->cgroups = init_task.cgroups;
+ cg = tsk->cgroups;
+ tsk->cgroups = &init_css_set;
  task_unlock(tsk);
+ if (cg)
+  put_css_set(cg);
 }

 /**
@@ -1848,7 +2284,7 @@ int cgroup_clone(struct task_struct *
```

```
   mutex_unlock(&cgroup_mutex);
   return 0;
  }
- cg = &tsk->cgroups;
+ cg = tsk->cgroups;
  parent = task_cgroup(tsk, subsys->subsys_id);

  snprintf(nodename, MAX_CGROUP_TYPE_NAMELEN, "node_%d", tsk->pid);
@@ -1856,6 +2292,8 @@ int cgroup_clone(struct task_struct *
  /* Pin the hierarchy */
  atomic_inc(&parent->root->sb->s_active);

+ /* Keep the cgroup alive */
+ get_css_set(cg);
  mutex_unlock(&cgroup_mutex);

  /* Now do the VFS work to create a cgroup */
@@ -1899,6 +2337,7 @@ int cgroup_clone(struct task_struct *
      (parent != task_cgroup(tsk, subsys->subsys_id))) {
   /* Aargh, we raced ... */
   mutex_unlock(&inode->i_mutex);
+  put_css_set(cg);

   deactivate_super(parent->root->sb);
   /* The cgroup is still accessible in the VFS, but
@@ -1922,6 +2361,7 @@ int cgroup_clone(struct task_struct *

  out_release:
  mutex_unlock(&inode->i_mutex);
+ put_css_set(cg);
  deactivate_super(parent->root->sb);
  return ret;
 }
diff -puN kernel/fork.c~task-cgroupsv11-shared-cgroup-subsystem-group-arrays kernel/fork.c
--- a/kernel/fork.c~task-cgroupsv11-shared-cgroup-subsystem-group-arrays
+++ a/kernel/fork.c
@@ -1289,6 +1289,7 @@ static struct task_struct *copy_process(
   put_user(p->pid, parent_tidptr);

  proc_fork_connector(p);
+ cgroup_post_fork(p);
  return p;

 bad_fork_cleanup_namespaces:
_

--
```

Subject: [PATCH 08/29] task containersv11 shared container subsystem group arrays avoid lockdep warning
Posted by Paul Menage on Tue, 11 Sep 2007 19:52:47 GMT
View Forum Message <> Reply to Message

From: Paul Menage <menage@google.com>

I think this is the right way to handle the lockdep false-positive in the current cgroups patches, but I'm not that familiar with lockdep so any suggestions for a better approach are welcomed.

In order to avoid a false-positive lockdep warning, we lock the root inode of a new filesystem mount prior to taking cgroup_mutex, to preserve the invariant that cgroup_mutex nests inside inode->i_mutex. In order to prevent a lockdep false positive when locking i_mutex on a newly-created cgroup directory inode we use mutex_lock_nested(), with a nesting level of I_MUTEX_CHILD since the new inode will ultimately be a child directory of the parent whose i_mutex is nested outside of cgroup_mutex.

Signed-off-by: Paul Menage <menage@google.com>
Acked-by: Peter Zijlstra <a.p.zijlstra@chello.nl>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

kernel/cgroup.c | 17 +++++++----------
1 files changed, 7 insertions(+), 10 deletions(-)

diff -puN kernel/cgroup.c~task-cgroupsv11-shared-cgroup-subsystem-group-arrays-avoid-lockdep-warning kernel/cgroup.c
---
a/kernel/cgroup.c~task-cgroupsv11-shared-cgroup-subsystem-group-arrays-avoid-lockdep-warning
+++ a/kernel/cgroup.c
@@ -867,13 +867,16 @@ static int cgroup_get_sb(struct file_
} else {
/* New superblock */
struct cgroup *cont = &root->top_cgroup;
+ struct inode *inode;

BUG_ON(sb->s_root != NULL);

ret = cgroup_get_rootdir(sb);

```
   if (ret)
    goto drop_new_super;
+   inode = sb->s_root->d_inode;

+   mutex_lock(&inode->i_mutex);
   mutex_lock(&cgroup_mutex);

   /*
@@ -886,12 +889,14 @@ static int cgroup_get_sb(struct file_
   ret = allocate_cg_links(css_set_count, &tmp_cg_links);
   if (ret) {
    mutex_unlock(&cgroup_mutex);
+    mutex_unlock(&inode->i_mutex);
    goto drop_new_super;
   }

   ret = rebind_subsystems(root, root->subsys_bits);
   if (ret == -EBUSY) {
    mutex_unlock(&cgroup_mutex);
+    mutex_unlock(&inode->i_mutex);
    goto drop_new_super;
   }

@@ -931,16 +936,8 @@ static int cgroup_get_sb(struct file_
   BUG_ON(!list_empty(&cont->children));
   BUG_ON(root->number_of_cgroups != 1);

-   /*
-    * I believe that it's safe to nest i_mutex inside
-    * cgroup_mutex in this case, since no-one else can
-    * be accessing this directory yet. But we still need
-    * to teach lockdep that this is the case - currently
-    * a cgroupfs remount triggers a lockdep warning
-    */
-   mutex_lock(&cont->dentry->d_inode->i_mutex);
   cgroup_populate_dir(cont);
-   mutex_unlock(&cont->dentry->d_inode->i_mutex);
+   mutex_unlock(&inode->i_mutex);
   mutex_unlock(&cgroup_mutex);
  }

@@ -1358,7 +1355,7 @@ static int cgroup_create_file(struct

   /* start with the directory inode held, so that we can
    * populate it without racing with another mkdir */
-   mutex_lock(&inode->i_mutex);
+   mutex_lock_nested(&inode->i_mutex, I_MUTEX_CHILD);
  } else if (S_ISREG(mode)) {
```

```
  inode->i_size = 0;
  inode->i_fop = &cgroup_file_operations;
```

–

--

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 09/29] task containersv11 shared container subsystem group arrays include fix

Posted by Paul Menage on Tue, 11 Sep 2007 19:52:48 GMT
View Forum Message <> Reply to Message

From: Andrew Morton <akpm@linux-foundation.org>

kernel/cgroup.c: In function 'cgroup_new_inode':
kernel/cgroup.c:573: error: variable 'cgroup_backing_dev_info' has initializer but incomplete type
kernel/cgroup.c:574: error: unknown field 'capabilities' specified in initializer
kernel/cgroup.c:574: error: 'BDI_CAP_NO_ACCT_DIRTY' undeclared (first use in this function)
kernel/cgroup.c:574: error: (Each undeclared identifier is reported only once)

Cc: Paul Menage <menage@google.com>
Cc: Peter Zijlstra <a.p.zijlstra@chello.nl>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

 kernel/cgroup.c | 1 +
 1 file changed, 1 insertion(+)

diff -puN kernel/cgroup.c~task-cgroupsv11-shared-cgroup-subsystem-group-arrays-include-fix kernel/cgroup.c
--- a/kernel/cgroup.c~task-cgroupsv11-shared-cgroup-subsystem-group-arrays-include-fix
+++ a/kernel/cgroup.c
@@ -36,6 +36,7 @@
 #include <linux/proc_fs.h>
 #include <linux/rcupdate.h>
 #include <linux/sched.h>
+#include <linux/backing-dev.h>
 #include <linux/seq_file.h>
 #include <linux/slab.h>
 #include <linux/magic.h>

–

--

Subject: [PATCH 10/29] task containersv11 automatic userspace notification of idle containers
Posted by Paul Menage on Tue, 11 Sep 2007 19:52:49 GMT
View Forum Message <> Reply to Message

From: Paul Menage <menage@google.com>

Add the following files to the cgroup filesystem:

notify_on_release - configures/reports whether the cgroup subsystem should attempt to run a release script when this cgroup becomes unused

release_agent - configures/reports the release agent to be used for this hierarchy (top level in each hierarchy only)

releasable - reports whether this cgroup would have been auto-released if notify_on_release was true and a release agent was configured (mainly useful for debugging)

To avoid locking issues, invoking the userspace release agent is done via a workqueue task; cgroups that need to have their release agents invoked by the workqueue task are linked on to a list.

Signed-off-by: Paul Menage <menage@google.com>
Cc: Serge E. Hallyn <serue@us.ibm.com>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Dave Hansen <haveblue@us.ibm.com>
Cc: Balbir Singh <balbir@in.ibm.com>
Cc: Paul Jackson <pj@sgi.com>
Cc: Kirill Korotaev <dev@openvz.org>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: Srivatsa Vaddagiri <vatsa@in.ibm.com>
Cc: Cedric Le Goater <clg@fr.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

include/linux/cgroup.h | 11
kernel/cgroup.c | 425 ++----
2 files changed, 393 insertions(+), 43 deletions(-)

diff -puN include/linux/cgroup.h~task-cgroupsv11-automatic-userspace-notification-of-idle-cgroups include/linux/cgroup.h

```
--- a/include/linux/cgroup.h~task-cgroupsv11-automatic-userspace-notification-of-idle-cgroups
+++ a/include/linux/cgroup.h
@@ -77,10 +77,11 @@ static inline void css_get(struct contai
  * css_get()
  */

+extern void __css_put(struct cgroup_subsys_state *css);
 static inline void css_put(struct cgroup_subsys_state *css)
 {
  if (!test_bit(CSS_ROOT, &css->flags))
-  atomic_dec(&css->refcnt);
+  __css_put(css);
 }

 struct cgroup {
@@ -112,6 +113,13 @@ struct cgroup {
   * tasks in this cgroup. Protected by css_set_lock
   */
  struct list_head css_sets;
+
+ /*
+  * Linked list running through all cgroups that can
+  * potentially be reaped by the release agent. Protected by
+  * release_list_lock
+  */
+ struct list_head release_list;
 };

 /* A css_set is a structure holding pointers to a set of
@@ -285,7 +293,6 @@ struct task_struct *cgroup_iter_next(
      struct cgroup_iter *it);
 void cgroup_iter_end(struct cgroup *cont, struct cgroup_iter *it);

-
 #else /* !CONFIG_CGROUPS */

 static inline int cgroup_init_early(void) { return 0; }
diff -puN kernel/cgroup.c~task-cgroupsv11-automatic-userspace-notification-of-idle-cgroups
kernel/cgroup.c
--- a/kernel/cgroup.c~task-cgroupsv11-automatic-userspace-notification-of-idle-cgroups
+++ a/kernel/cgroup.c
@@ -44,6 +44,8 @@
 #include <linux/sort.h>
 #include <asm/atomic.h>

+static DEFINE_MUTEX(cgroup_mutex);
+
 /* Generate an array of cgroup subsystem pointers */
```

```
 #define SUBSYS(_x) &_x ## _subsys,

@@ -82,6 +84,13 @@ struct cgroupfs_root {

  /* Hierarchy-specific flags */
  unsigned long flags;
+
+ /* The path to use for release notifications. No locking
+  * between setting and use - so if userspace updates this
+  * while child cgroups exist, you could miss a
+  * notification. We ensure that it's always a valid
+  * NUL-terminated string */
+ char release_agent_path[PATH_MAX];
 };


@@ -109,7 +118,13 @@ static int need_forkexit_callback;

 /* bits in struct cgroup flags field */
 enum {
+ /* Control Group is dead */
  CONT_REMOVED,
+ /* Control Group has previously had a child cgroup or a task,
+  * but no longer (only if CONT_NOTIFY_ON_RELEASE is set) */
+ CONT_RELEASABLE,
+ /* Control Group requires release notifications to userspace */
+ CONT_NOTIFY_ON_RELEASE,
 };

 /* convenient tests for these bits */
@@ -123,6 +138,19 @@ enum {
  ROOT_NOPREFIX, /* mounted subsystems have no named prefix */
 };

+inline int cgroup_is_releasable(const struct cgroup *cont)
+{
+ const int bits =
+  (1 << CONT_RELEASABLE) |
+  (1 << CONT_NOTIFY_ON_RELEASE);
+ return (cont->flags & bits) == bits;
+}
+
+inline int notify_on_release(const struct cgroup *cont)
+{
+ return test_bit(CONT_NOTIFY_ON_RELEASE, &cont->flags);
+}
+
 /*
```

```
  * for_each_subsys() allows you to iterate on each subsystem attached to
  * an active hierarchy
@@ -134,6 +162,14 @@ list_for_each_entry(_ss, &_root->subsys_
 #define for_each_root(_root) \
 list_for_each_entry(_root, &roots, root_list)

+/* the list of cgroups eligible for automatic release. Protected by
+ * release_list_lock */
+static LIST_HEAD(release_list);
+static DEFINE_SPINLOCK(release_list_lock);
+static void cgroup_release_agent(struct work_struct *work);
+static DECLARE_WORK(release_agent_work, cgroup_release_agent);
+static void check_for_release(struct cgroup *cont);
+
 /* Link structure for associating css_set objects with cgroups */
 struct cg_cgroup_link {
  /*
@@ -188,11 +224,8 @@ static int use_task_css_set_links;
 /*
  * unlink a css_set from the list and free it
  */
-static void release_css_set(struct kref *k)
+static void unlink_css_set(struct css_set *cg)
 {
- struct css_set *cg = container_of(k, struct css_set, ref);
- int i;
-
  write_lock(&css_set_lock);
  list_del(&cg->list);
  css_set_count--;
@@ -205,11 +238,39 @@ static void release_css_set(struct kre
   kfree(link);
  }
  write_unlock(&css_set_lock);
- for (i = 0; i < CGROUP_SUBSYS_COUNT; i++)
-  atomic_dec(&cg->subsys[i]->cgroup->count);
+}
+
+static void __release_css_set(struct kref *k, int taskexit)
+{
+ int i;
+ struct css_set *cg = container_of(k, struct css_set, ref);
+
+ unlink_css_set(cg);
+
+ rcu_read_lock();
+ for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
+  struct cgroup *cont = cg->subsys[i]->cgroup;
```

```
+  if (atomic_dec_and_test(&cont->count) &&
+      notify_on_release(cont)) {
+   if (taskexit)
+    set_bit(CONT_RELEASABLE, &cont->flags);
+   check_for_release(cont);
+  }
+ }
+ rcu_read_unlock();
  kfree(cg);
 }

+static void release_css_set(struct kref *k)
+{
+ __release_css_set(k, 0);
+}
+
+static void release_css_set_taskexit(struct kref *k)
+{
+ __release_css_set(k, 1);
+}
+
 /*
  * refcounted get/put for css_set objects
  */
@@ -223,6 +284,11 @@ static inline void put_css_set(struct
  kref_put(&cg->ref, release_css_set);
 }

+static inline void put_css_set_taskexit(struct css_set *cg)
+{
+ kref_put(&cg->ref, release_css_set_taskexit);
+}
+
 /*
  * find_existing_css_set() is a helper for
  * find_css_set(), and checks to see whether an existing
@@ -464,8 +530,6 @@ static struct css_set *find_css_set(
  * update of a tasks cgroup pointer by attach_task()
  */

-static DEFINE_MUTEX(cgroup_mutex);
-
 /**
  * cgroup_lock - lock out any changes to cgroup structures
  *
@@ -524,6 +588,13 @@ static void cgroup_diput(struct dentr
  if (S_ISDIR(inode->i_mode)) {
   struct cgroup *cont = dentry->d_fsdata;
```

```
   BUG_ON(!(cgroup_is_removed(cont)));
+  /* It's possible for external users to be holding css
+   * reference counts on a cgroup; css_put() needs to
+   * be able to access the cgroup after decrementing
+   * the reference count in order to know if it needs to
+   * queue the cgroup to be handled by the release
+   * agent */
+  synchronize_rcu();
   kfree(cont);
  }
  iput(inode);
@@ -668,6 +739,8 @@ static int cgroup_show_options(struct
   seq_printf(seq, ",%s", ss->name);
  if (test_bit(ROOT_NOPREFIX, &root->flags))
   seq_puts(seq, ",noprefix");
+ if (strlen(root->release_agent_path))
+  seq_printf(seq, ",release_agent=%s", root->release_agent_path);
  mutex_unlock(&cgroup_mutex);
  return 0;
 }
@@ -675,6 +748,7 @@ static int cgroup_show_options(struct
 struct cgroup_sb_opts {
  unsigned long subsys_bits;
  unsigned long flags;
+ char *release_agent;
 };

 /* Convert a hierarchy specifier into a bitmask of subsystems and
@@ -686,6 +760,7 @@ static int parse_cgroupfs_options(cha

  opts->subsys_bits = 0;
  opts->flags = 0;
+ opts->release_agent = NULL;

  while ((token = strsep(&o, ",")) != NULL) {
   if (!*token)
@@ -694,6 +769,15 @@ static int parse_cgroupfs_options(cha
    opts->subsys_bits = (1 << CGROUP_SUBSYS_COUNT) - 1;
   } else if (!strcmp(token, "noprefix")) {
    set_bit(ROOT_NOPREFIX, &opts->flags);
+  } else if (!strncmp(token, "release_agent=", 14)) {
+   /* Specifying two release agents is forbidden */
+   if (opts->release_agent)
+    return -EINVAL;
+   opts->release_agent = kzalloc(PATH_MAX, GFP_KERNEL);
+   if (!opts->release_agent)
+    return -ENOMEM;
+   strncpy(opts->release_agent, token + 14, PATH_MAX - 1);
```

```
+   opts->release_agent[PATH_MAX - 1] = 0;
   } else {
    struct cgroup_subsys *ss;
    int i;
@@ -743,7 +827,11 @@ static int cgroup_remount(struct supe
  if (!ret)
   cgroup_populate_dir(cont);

+ if (opts.release_agent)
+  strcpy(root->release_agent_path, opts.release_agent);
  out_unlock:
+ if (opts.release_agent)
+  kfree(opts.release_agent);
  mutex_unlock(&cgroup_mutex);
  mutex_unlock(&cont->dentry->d_inode->i_mutex);
  return ret;
@@ -767,6 +855,7 @@ static void init_cgroup_root(struct c
  INIT_LIST_HEAD(&cont->sibling);
  INIT_LIST_HEAD(&cont->children);
  INIT_LIST_HEAD(&cont->css_sets);
+ INIT_LIST_HEAD(&cont->release_list);
 }

 static int cgroup_test_super(struct super_block *sb, void *data)
@@ -841,8 +930,11 @@ static int cgroup_get_sb(struct file_

  /* First find the desired set of subsystems */
  ret = parse_cgroupfs_options(data, &opts);
- if (ret)
+ if (ret) {
+  if (opts.release_agent)
+   kfree(opts.release_agent);
   return ret;
+ }

  root = kzalloc(sizeof(*root), GFP_KERNEL);
  if (!root)
@@ -851,6 +943,10 @@ static int cgroup_get_sb(struct file_
  init_cgroup_root(root);
  root->subsys_bits = opts.subsys_bits;
  root->flags = opts.flags;
+ if (opts.release_agent) {
+  strcpy(root->release_agent_path, opts.release_agent);
+  kfree(opts.release_agent);
+ }

  sb = sget(fs_type, cgroup_test_super, cgroup_set_super, root);
```

```
@@ -1131,7 +1227,7 @@ static int attach_task(struct cgroup
    ss->attach(ss, cont, oldcont, tsk);
   }
  }
-
+ set_bit(CONT_RELEASABLE, &oldcont->flags);
  synchronize_rcu();
  put_css_set(cg);
  return 0;
@@ -1181,6 +1277,9 @@ enum cgroup_filetype {
  FILE_ROOT,
  FILE_DIR,
  FILE_TASKLIST,
+ FILE_NOTIFY_ON_RELEASE,
+ FILE_RELEASABLE,
+ FILE_RELEASE_AGENT,
 };

 static ssize_t cgroup_common_file_write(struct cgroup *cont,
@@ -1218,6 +1317,32 @@ static ssize_t cgroup_common_file_wri
  case FILE_TASKLIST:
   retval = attach_task_by_pid(cont, buffer);
   break;
+ case FILE_NOTIFY_ON_RELEASE:
+  clear_bit(CONT_RELEASABLE, &cont->flags);
+  if (simple_strtoul(buffer, NULL, 10) != 0)
+   set_bit(CONT_NOTIFY_ON_RELEASE, &cont->flags);
+  else
+   clear_bit(CONT_NOTIFY_ON_RELEASE, &cont->flags);
+  break;
+ case FILE_RELEASE_AGENT:
+ {
+  struct cgroupfs_root *root = cont->root;
+  /* Strip trailing newline */
+  if (nbytes && (buffer[nbytes-1] == '\n')) {
+   buffer[nbytes-1] = 0;
+  }
+  if (nbytes < sizeof(root->release_agent_path)) {
+   /* We never write anything other than '\0'
+    * into the last char of release_agent_path,
+    * so it always remains a NUL-terminated
+    * string */
+   strncpy(root->release_agent_path, buffer, nbytes);
+   root->release_agent_path[nbytes] = 0;
+  } else {
+   retval = -ENOSPC;
+  }
+  break;
```

```
+ }
  default:
   retval = -EINVAL;
   goto out2;
@@ -1258,6 +1383,49 @@ static ssize_t cgroup_read_uint(struc
  return simple_read_from_buffer(buf, nbytes, ppos, tmp, len);
 }

+static ssize_t cgroup_common_file_read(struct cgroup *cont,
+       struct cftype *cft,
+       struct file *file,
+       char __user *buf,
+       size_t nbytes, loff_t *ppos)
+{
+ enum cgroup_filetype type = cft->private;
+ char *page;
+ ssize_t retval = 0;
+ char *s;
+
+ if (!(page = (char *)__get_free_page(GFP_KERNEL)))
+  return -ENOMEM;
+
+ s = page;
+
+ switch (type) {
+ case FILE_RELEASE_AGENT:
+ {
+  struct cgroupfs_root *root;
+  size_t n;
+  mutex_lock(&cgroup_mutex);
+  root = cont->root;
+  n = strnlen(root->release_agent_path,
+       sizeof(root->release_agent_path));
+  n = min(n, (size_t) PAGE_SIZE);
+  strncpy(s, root->release_agent_path, n);
+  mutex_unlock(&cgroup_mutex);
+  s += n;
+  break;
+ }
+ default:
+  retval = -EINVAL;
+  goto out;
+ }
+ *s++ = '\n';
+
+ retval = simple_read_from_buffer(buf, nbytes, ppos, page, s - page);
+out:
+ free_page((unsigned long)page);
```

```
+ return retval;
+}
+
 static ssize_t cgroup_file_read(struct file *file, char __user *buf,
        size_t nbytes, loff_t *ppos)
 {
@@ -1673,16 +1841,49 @@ static int cgroup_tasks_release(struc
  return 0;
 }

+static u64 cgroup_read_notify_on_release(struct cgroup *cont,
+         struct cftype *cft)
+{
+ return notify_on_release(cont);
+}
+
+static u64 cgroup_read_releasable(struct cgroup *cont, struct cftype *cft)
+{
+ return test_bit(CONT_RELEASABLE, &cont->flags);
+}
+
 /*
  * for the common functions, 'private' gives the type of file
  */
-static struct cftype cft_tasks = {
- .name = "tasks",
- .open = cgroup_tasks_open,
- .read = cgroup_tasks_read,
+static struct cftype files[] = {
+ {
+  .name = "tasks",
+  .open = cgroup_tasks_open,
+  .read = cgroup_tasks_read,
+  .write = cgroup_common_file_write,
+  .release = cgroup_tasks_release,
+  .private = FILE_TASKLIST,
+ },
+
+ {
+  .name = "notify_on_release",
+  .read_uint = cgroup_read_notify_on_release,
+  .write = cgroup_common_file_write,
+  .private = FILE_NOTIFY_ON_RELEASE,
+ },
+
+ {
+  .name = "releasable",
+  .read_uint = cgroup_read_releasable,
```

```
+  .private = FILE_RELEASABLE,
+ }
+};
+
+static struct cftype cft_release_agent = {
+ .name = "release_agent",
+ .read = cgroup_common_file_read,
  .write = cgroup_common_file_write,
- .release = cgroup_tasks_release,
- .private = FILE_TASKLIST,
+ .private = FILE_RELEASE_AGENT,
 };

 static int cgroup_populate_dir(struct cgroup *cont)
@@ -1693,10 +1894,15 @@ static int cgroup_populate_dir(struct
  /* First clear out any existing files */
  cgroup_clear_directory(cont->dentry);

- err = cgroup_add_file(cont, NULL, &cft_tasks);
+ err = cgroup_add_files(cont, NULL, files, ARRAY_SIZE(files));
  if (err < 0)
   return err;

+ if (cont == cont->top_cgroup) {
+  if ((err = cgroup_add_file(cont, NULL, &cft_release_agent)) < 0)
+   return err;
+ }
+
  for_each_subsys(cont->root, ss) {
   if (ss->populate && (err = ss->populate(ss, cont)) < 0)
    return err;
@@ -1753,6 +1959,7 @@ static long cgroup_create(struct cont
  INIT_LIST_HEAD(&cont->sibling);
  INIT_LIST_HEAD(&cont->children);
  INIT_LIST_HEAD(&cont->css_sets);
+ INIT_LIST_HEAD(&cont->release_list);

  cont->parent = parent;
  cont->root = parent->root;
@@ -1814,6 +2021,38 @@ static int cgroup_mkdir(struct inode
  return cgroup_create(c_parent, dentry, mode | S_IFDIR);
 }

+static inline int cgroup_has_css_refs(struct cgroup *cont)
+{
+ /* Check the reference count on each subsystem. Since we
+  * already established that there are no tasks in the
+  * cgroup, if the css refcount is also 0, then there should
```

```
+  * be no outstanding references, so the subsystem is safe to
+  * destroy. We scan across all subsystems rather than using
+  * the per-hierarchy linked list of mounted subsystems since
+  * we can be called via check_for_release() with no
+  * synchronization other than RCU, and the subsystem linked
+  * list isn't RCU-safe */
+ int i;
+ for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
+  struct cgroup_subsys *ss = subsys[i];
+  struct cgroup_subsys_state *css;
+  /* Skip subsystems not in this hierarchy */
+  if (ss->root != cont->root)
+   continue;
+  css = cont->subsys[ss->subsys_id];
+  /* When called from check_for_release() it's possible
+   * that by this point the cgroup has been removed
+   * and the css deleted. But a false-positive doesn't
+   * matter, since it can only happen if the cgroup
+   * has been deleted and hence no longer needs the
+   * release agent to be called anyway. */
+  if (css && atomic_read(&css->refcnt)) {
+   return 1;
+  }
+ }
+ return 0;
+}
+
 static int cgroup_rmdir(struct inode *unused_dir, struct dentry *dentry)
 {
  struct cgroup *cont = dentry->d_fsdata;
@@ -1822,7 +2061,6 @@ static int cgroup_rmdir(struct inode
  struct cgroup_subsys *ss;
  struct super_block *sb;
  struct cgroupfs_root *root;
- int css_busy = 0;

  /* the vfs holds both inode->i_mutex already */

@@ -1840,20 +2078,7 @@ static int cgroup_rmdir(struct inode
  root = cont->root;
  sb = root->sb;

- /* Check the reference count on each subsystem. Since we
-  * already established that there are no tasks in the
-  * cgroup, if the css refcount is also 0, then there should
-  * be no outstanding references, so the subsystem is safe to
-  * destroy */
- for_each_subsys(root, ss) {
```

```
-  struct cgroup_subsys_state *css;
-  css = cont->subsys[ss->subsys_id];
-  if (atomic_read(&css->refcnt)) {
-   css_busy = 1;
-   break;
-  }
- }
- if (css_busy) {
+ if (cgroup_has_css_refs(cont)) {
   mutex_unlock(&cgroup_mutex);
   return -EBUSY;
  }
@@ -1863,7 +2088,11 @@ static int cgroup_rmdir(struct inode
    ss->destroy(ss, cont);
  }

+ spin_lock(&release_list_lock);
  set_bit(CONT_REMOVED, &cont->flags);
+ if (!list_empty(&cont->release_list))
+  list_del(&cont->release_list);
+ spin_unlock(&release_list_lock);
  /* delete my sibling from parent->children */
  list_del(&cont->sibling);
  spin_lock(&cont->dentry->d_lock);
@@ -1875,6 +2104,9 @@ static int cgroup_rmdir(struct inode
  dput(d);
  root->number_of_cgroups--;

+ set_bit(CONT_RELEASABLE, &parent->flags);
+ check_for_release(parent);
+
  mutex_unlock(&cgroup_mutex);
  /* Drop the active superblock reference that we took when we
   * created the cgroup */
@@ -1912,15 +2144,15 @@ static void cgroup_init_subsys(struct
   /* If this subsystem requested that it be notified with fork
    * events, we should send it one now for every process in the
    * system */
-  if (ss->fork) {
-   struct task_struct *g, *p;
+ if (ss->fork) {
+  struct task_struct *g, *p;

-   read_lock(&tasklist_lock);
-   do_each_thread(g, p) {
-    ss->fork(ss, p);
-   } while_each_thread(g, p);
-   read_unlock(&tasklist_lock);
```

```
-  }
+  read_lock(&tasklist_lock);
+  do_each_thread(g, p) {
+   ss->fork(ss, p);
+  } while_each_thread(g, p);
+  read_unlock(&tasklist_lock);
+ }

  need_forkexit_callback |= ss->fork || ss->exit;

@@ -2247,7 +2479,7 @@ void cgroup_exit(struct task_struct *
  tsk->cgroups = &init_css_set;
  task_unlock(tsk);
  if (cg)
-  put_css_set(cg);
+  put_css_set_taskexit(cg);
 }

 /**
@@ -2358,7 +2590,10 @@ int cgroup_clone(struct task_struct *

  out_release:
  mutex_unlock(&inode->i_mutex);
+
+ mutex_lock(&cgroup_mutex);
  put_css_set(cg);
+ mutex_unlock(&cgroup_mutex);
  deactivate_super(parent->root->sb);
  return ret;
 }
@@ -2388,3 +2623,111 @@ int cgroup_is_descendant(const struct
  ret = (cont == target);
  return ret;
 }
+
+static void check_for_release(struct cgroup *cont)
+{
+ /* All of these checks rely on RCU to keep the cgroup
+  * structure alive */
+ if (cgroup_is_releasable(cont) && !atomic_read(&cont->count)
+     && list_empty(&cont->children) && !cgroup_has_css_refs(cont)) {
+  /* Control Group is currently removeable. If it's not
+   * already queued for a userspace notification, queue
+   * it now */
+  int need_schedule_work = 0;
+  spin_lock(&release_list_lock);
+  if (!cgroup_is_removed(cont) &&
+      list_empty(&cont->release_list)) {
```

```
+   list_add(&cont->release_list, &release_list);
+   need_schedule_work = 1;
+  }
+  spin_unlock(&release_list_lock);
+  if (need_schedule_work)
+   schedule_work(&release_agent_work);
+ }
+}
+
+void __css_put(struct cgroup_subsys_state *css)
+{
+ struct cgroup *cont = css->cgroup;
+ rcu_read_lock();
+ if (atomic_dec_and_test(&css->refcnt) && notify_on_release(cont)) {
+  set_bit(CONT_RELEASABLE, &cont->flags);
+  check_for_release(cont);
+ }
+ rcu_read_unlock();
+}
+
+/*
+ * Notify userspace when a cgroup is released, by running the
+ * configured release agent with the name of the cgroup (path
+ * relative to the root of cgroup file system) as the argument.
+ *
+ * Most likely, this user command will try to rmdir this cgroup.
+ *
+ * This races with the possibility that some other task will be
+ * attached to this cgroup before it is removed, or that some other
+ * user task will 'mkdir' a child cgroup of this cgroup.  That's ok.
+ * The presumed 'rmdir' will fail quietly if this cgroup is no longer
+ * unused, and this cgroup will be reprieved from its death sentence,
+ * to continue to serve a useful existence.  Next time it's released,
+ * we will get notified again, if it still has 'notify_on_release' set.
+ *
+ * The final arg to call_usermodehelper() is UMH_WAIT_EXEC, which
+ * means only wait until the task is successfully execve()'d.  The
+ * separate release agent task is forked by call_usermodehelper(),
+ * then control in this thread returns here, without waiting for the
+ * release agent task.  We don't bother to wait because the caller of
+ * this routine has no use for the exit status of the release agent
+ * task, so no sense holding our caller up for that.
+ *
+ */
+
+static void cgroup_release_agent(struct work_struct *work)
+{
+ BUG_ON(work != &release_agent_work);
```

```
+ mutex_lock(&cgroup_mutex);
+ spin_lock(&release_list_lock);
+ while (!list_empty(&release_list)) {
+  char *argv[3], *envp[3];
+  int i;
+  char *pathbuf;
+  struct cgroup *cont = list_entry(release_list.next,
+         struct cgroup,
+         release_list);
+  list_del_init(&cont->release_list);
+  spin_unlock(&release_list_lock);
+  pathbuf = kmalloc(PAGE_SIZE, GFP_KERNEL);
+  if (!pathbuf) {
+   spin_lock(&release_list_lock);
+   continue;
+  }
+
+  if (cgroup_path(cont, pathbuf, PAGE_SIZE) < 0) {
+   kfree(pathbuf);
+   spin_lock(&release_list_lock);
+   continue;
+  }
+
+  i = 0;
+  argv[i++] = cont->root->release_agent_path;
+  argv[i++] = (char *)pathbuf;
+  argv[i] = NULL;
+
+  i = 0;
+  /* minimal command environment */
+  envp[i++] = "HOME=/";
+  envp[i++] = "PATH=/sbin:/bin:/usr/sbin:/usr/bin";
+  envp[i] = NULL;
+
+  /* Drop the lock while we invoke the usermode helper,
+   * since the exec could involve hitting disk and hence
+   * be a slow process */
+  mutex_unlock(&cgroup_mutex);
+  call_usermodehelper(argv[0], argv, envp, UMH_WAIT_EXEC);
+  kfree(pathbuf);
+  mutex_lock(&cgroup_mutex);
+  spin_lock(&release_list_lock);
+ }
+ spin_unlock(&release_list_lock);
+ mutex_unlock(&cgroup_mutex);
+}
_
```

--

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH 11/29] task containersv11 make cpusets a client of containers
Posted by Paul Menage on Tue, 11 Sep 2007 19:52:50 GMT
View Forum Message <> Reply to Message

From: Paul Menage <menage@google.com>

Remove the filesystem support logic from the cpusets system and makes cpusets a cgroup subsystem

The "cpuset" filesystem becomes a dummy filesystem; attempts to mount it get passed through to the cgroup filesystem with the appropriate options to emulate the old cpuset filesystem behaviour.

Signed-off-by: Paul Menage <menage@google.com>
Cc: Serge E. Hallyn <serue@us.ibm.com>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Dave Hansen <haveblue@us.ibm.com>
Cc: Balbir Singh <balbir@in.ibm.com>
Cc: Paul Jackson <pj@sgi.com>
Cc: Kirill Korotaev <dev@openvz.org>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: Srivatsa Vaddagiri <vatsa@in.ibm.com>
Cc: Cedric Le Goater <clg@fr.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>

```
 Documentation/cpusets.txt       |   91 +-
 fs/proc/base.c                  |    4
 include/linux/cgroup_subsys.h   |    6
 include/linux/cpuset.h          |   12
 include/linux/mempolicy.h       |   12
 include/linux/sched.h           |    3
 init/Kconfig                    |    7
 kernel/cpuset.c                 | 1192 +++++------------------
 kernel/exit.c                   |    2
 kernel/fork.c                   |    3
 mm/mempolicy.c                  |    2
 11 files changed, 278 insertions(+), 1056 deletions(-)
```

diff -puN Documentation/cpusets.txt~task-cgroupsv11-make-cpusets-a-client-of-cgroups Documentation/cpusets.txt

```
--- a/Documentation/cpusets.txt~task-cgroupsv11-make-cpusets-a-client-of-cgroups
+++ a/Documentation/cpusets.txt
@@ -7,6 +7,7 @@ Written by Simon.Derr@bull.net
 Portions Copyright (c) 2004-2006 Silicon Graphics, Inc.
 Modified by Paul Jackson <pj@sgi.com>
 Modified by Christoph Lameter <clameter@sgi.com>
+Modified by Paul Menage <menage@google.com>

 CONTENTS:
 =========
@@ -16,10 +17,9 @@ CONTENTS:
   1.2 Why are cpusets needed ?
   1.3 How are cpusets implemented ?
   1.4 What are exclusive cpusets ?
-  1.5 What does notify_on_release do ?
-  1.6 What is memory_pressure ?
-  1.7 What is memory spread ?
-  1.8 How do I use cpusets ?
+  1.5 What is memory_pressure ?
+  1.6 What is memory spread ?
+  1.7 How do I use cpusets ?
 2. Usage Examples and Syntax
   2.1 Basic Usage
   2.2 Adding/removing cpus
@@ -44,18 +44,19 @@ hierarchy visible in a virtual file syst
 hooks, beyond what is already present, required to manage dynamic
 job placement on large systems.

-Each task has a pointer to a cpuset.  Multiple tasks may reference
-the same cpuset.  Requests by a task, using the sched_setaffinity(2)
-system call to include CPUs in its CPU affinity mask, and using the
-mbind(2) and set_mempolicy(2) system calls to include Memory Nodes
-in its memory policy, are both filtered through that tasks cpuset,
-filtering out any CPUs or Memory Nodes not in that cpuset.  The
-scheduler will not schedule a task on a CPU that is not allowed in
-its cpus_allowed vector, and the kernel page allocator will not
-allocate a page on a node that is not allowed in the requesting tasks
-mems_allowed vector.
+Cpusets use the generic cgroup subsystem described in
+Documentation/cgroup.txt.

-User level code may create and destroy cpusets by name in the cpuset
+Requests by a task, using the sched_setaffinity(2) system call to
+include CPUs in its CPU affinity mask, and using the mbind(2) and
+set_mempolicy(2) system calls to include Memory Nodes in its memory
+policy, are both filtered through that tasks cpuset, filtering out any
+CPUs or Memory Nodes not in that cpuset.  The scheduler will not
+schedule a task on a CPU that is not allowed in its cpus_allowed
```

```
+vector, and the kernel page allocator will not allocate a page on a
+node that is not allowed in the requesting tasks mems_allowed vector.
+
+User level code may create and destroy cpusets by name in the cgroup
 virtual file system, manage the attributes and permissions of these
 cpusets and which CPUs and Memory Nodes are assigned to each cpuset,
 specify and query to which cpuset a task is assigned, and list the
@@ -115,7 +116,7 @@ Cpusets extends these two mechanisms as
   - Cpusets are sets of allowed CPUs and Memory Nodes, known to the
     kernel.
   - Each task in the system is attached to a cpuset, via a pointer
-  in the task structure to a reference counted cpuset structure.
+   in the task structure to a reference counted cgroup structure.
   - Calls to sched_setaffinity are filtered to just those CPUs
     allowed in that tasks cpuset.
   - Calls to mbind and set_mempolicy are filtered to just
@@ -145,15 +146,10 @@ into the rest of the kernel, none in per
   - in page_alloc.c, to restrict memory to allowed nodes.
   - in vmscan.c, to restrict page recovery to the current cpuset.

-In addition a new file system, of type "cpuset" may be mounted,
-typically at /dev/cpuset, to enable browsing and modifying the cpusets
-presently known to the kernel.  No new system calls are added for
-cpusets - all support for querying and modifying cpusets is via
-this cpuset file system.
-
-Each task under /proc has an added file named 'cpuset', displaying
-the cpuset name, as the path relative to the root of the cpuset file
-system.
+You should mount the "cgroup" filesystem type in order to enable
+browsing and modifying the cpusets presently known to the kernel.  No
+new system calls are added for cpusets - all support for querying and
+modifying cpusets is via this cpuset file system.

 The /proc/<pid>/status file for each task has two added lines,
 displaying the tasks cpus_allowed (on which CPUs it may be scheduled)
@@ -163,16 +159,15 @@ in the format seen in the following exam
   Mems_allowed:   ffffffff,ffffffff

-Each cpuset is represented by a directory in the cpuset file system
-containing the following files describing that cpuset:
+Each cpuset is represented by a directory in the cgroup file system
+containing (on top of the standard cgroup files) the following
+files describing that cpuset:

   - cpus: list of CPUs in that cpuset
   - mems: list of Memory Nodes in that cpuset
```

  - memory_migrate flag: if set, move pages to cpusets nodes
  - cpu_exclusive flag: is cpu placement exclusive?
  - mem_exclusive flag: is memory placement exclusive?
- - tasks: list of tasks (by pid) attached to that cpuset
- - notify_on_release flag: run /sbin/cpuset_release_agent on exit?
  - memory_pressure: measure of how much paging pressure in cpuset

 In addition, the root cpuset only has the following file:
@@ -237,21 +232,7 @@ such as requests from interrupt handlers
 outside even a mem_exclusive cpuset.

-1.5 What does notify_on_release do ?

-
-If the notify_on_release flag is enabled (1) in a cpuset, then whenever
-the last task in the cpuset leaves (exits or attaches to some other
-cpuset) and the last child cpuset of that cpuset is removed, then
-the kernel runs the command /sbin/cpuset_release_agent, supplying the
-pathname (relative to the mount point of the cpuset file system) of the
-abandoned cpuset. This enables automatic removal of abandoned cpusets.
-The default value of notify_on_release in the root cpuset at system
-boot is disabled (0). The default value of other cpusets at creation
-is the current value of their parents notify_on_release setting.
-
-
-1.6 What is memory_pressure ?
+1.5 What is memory_pressure ?

 The memory_pressure of a cpuset provides a simple per-cpuset metric
 of the rate that the tasks in a cpuset are attempting to free up in
@@ -308,7 +289,7 @@ the tasks in the cpuset, in units of rec
 times 1000.

-1.7 What is memory spread ?
+1.6 What is memory spread ?

 There are two boolean flag files per cpuset that control where the
 kernel allocates pages for the file system buffers and related in
@@ -379,7 +360,7 @@ data set, the memory allocation across t
 can become very uneven.

-1.8 How do I use cpusets ?
+1.7 How do I use cpusets ?

```
 In order to minimize the impact of cpusets on critical kernel
@@ -469,7 +450,7 @@ than stress the kernel.
 To start a new job that is to be contained within a cpuset, the steps are:

  1) mkdir /dev/cpuset
- 2) mount -t cpuset none /dev/cpuset
+ 2) mount -t cgroup -ocpuset cpuset /dev/cpuset
  3) Create the new cpuset by doing mkdir's and write's (or echo's) in
     the /dev/cpuset virtual file system.
  4) Start a task that will be the "founding father" of the new job.
@@ -481,7 +462,7 @@ For example, the following sequence of c
 named "Charlie", containing just CPUs 2 and 3, and Memory Node 1,
 and then start a subshell 'sh' in that cpuset:

-  mount -t cpuset none /dev/cpuset
+  mount -t cgroup -ocpuset cpuset /dev/cpuset
   cd /dev/cpuset
   mkdir Charlie
   cd Charlie
@@ -513,7 +494,7 @@ Creating, modifying, using the cpusets c
 virtual filesystem.

 To mount it, type:
-# mount -t cpuset none /dev/cpuset
+# mount -t cgroup -o cpuset cpuset /dev/cpuset

 Then under /dev/cpuset you can find a tree that corresponds to the
 tree of the cpusets in the system. For instance, /dev/cpuset
@@ -556,6 +537,18 @@ To remove a cpuset, just use rmdir:
 This will fail if the cpuset is in use (has cpusets inside, or has
 processes attached).

+Note that for legacy reasons, the "cpuset" filesystem exists as a
+wrapper around the cgroup filesystem.
+
+The command
+
+mount -t cpuset X /dev/cpuset
+
+is equivalent to
+
+mount -t cgroup -ocpuset X /dev/cpuset
+echo "/sbin/cpuset_release_agent" > /dev/cpuset/release_agent
+
 2.2 Adding/removing cpus
 ------------------------

diff -puN fs/proc/base.c~task-cgroupsv11-make-cpusets-a-client-of-cgroups fs/proc/base.c
```

```
--- a/fs/proc/base.c~task-cgroupsv11-make-cpusets-a-client-of-cgroups
+++ a/fs/proc/base.c
@@ -2049,7 +2049,7 @@ static const struct pid_entry tgid_base_
 #ifdef CONFIG_SCHEDSTATS
   INF("schedstat",  S_IRUGO, pid_schedstat),
 #endif
-#ifdef CONFIG_CPUSETS
+#ifdef CONFIG_PROC_PID_CPUSET
   REG("cpuset",     S_IRUGO, cpuset),
 #endif
 #ifdef CONFIG_CGROUPS
@@ -2341,7 +2341,7 @@ static const struct pid_entry tid_base_s
 #ifdef CONFIG_SCHEDSTATS
   INF("schedstat", S_IRUGO, pid_schedstat),
 #endif
-#ifdef CONFIG_CPUSETS
+#ifdef CONFIG_PROC_PID_CPUSET
   REG("cpuset",    S_IRUGO, cpuset),
 #endif
 #ifdef CONFIG_CGROUPS
diff -puN include/linux/cgroup_subsys.h~task-cgroupsv11-make-cpusets-a-client-of-cgroups
include/linux/cgroup_subsys.h
--- a/include/linux/cgroup_subsys.h~task-cgroupsv11-make-cpusets-a-client-of-cgroups
+++ a/include/linux/cgroup_subsys.h
@@ -7,4 +7,10 @@

 /* */

+#ifdef CONFIG_CPUSETS
+SUBSYS(cpuset)
+#endif
+
+/* */
+
 /* */
diff -puN include/linux/cpuset.h~task-cgroupsv11-make-cpusets-a-client-of-cgroups
include/linux/cpuset.h
--- a/include/linux/cpuset.h~task-cgroupsv11-make-cpusets-a-client-of-cgroups
+++ a/include/linux/cpuset.h
@@ -11,6 +11,7 @@
 #include <linux/sched.h>
 #include <linux/cpumask.h>
 #include <linux/nodemask.h>
+#include <linux/cgroup.h>

 #ifdef CONFIG_CPUSETS

@@ -19,8 +20,6 @@ extern int number_of_cpusets; /* How man
```

```
 extern int cpuset_init_early(void);
 extern int cpuset_init(void);
 extern void cpuset_init_smp(void);
-extern void cpuset_fork(struct task_struct *p);
-extern void cpuset_exit(struct task_struct *p);
 extern cpumask_t cpuset_cpus_allowed(struct task_struct *p);
 extern nodemask_t cpuset_mems_allowed(struct task_struct *p);
 #define cpuset_current_mems_allowed (current->mems_allowed)
@@ -75,13 +74,13 @@ static inline int cpuset_do_slab_mem_spr

 extern void cpuset_track_online_nodes(void);

+extern int current_cpuset_is_being_rebound(void);
+
 #else /* !CONFIG_CPUSETS */

 static inline int cpuset_init_early(void) { return 0; }
 static inline int cpuset_init(void) { return 0; }
 static inline void cpuset_init_smp(void) {}
-static inline void cpuset_fork(struct task_struct *p) {}
-static inline void cpuset_exit(struct task_struct *p) {}

 static inline cpumask_t cpuset_cpus_allowed(struct task_struct *p)
 {
@@ -146,6 +145,11 @@ static inline int cpuset_do_slab_mem_spr

 static inline void cpuset_track_online_nodes(void) {}

+static inline int current_cpuset_is_being_rebound(void)
+{
+ return 0;
+}
+
 #endif /* !CONFIG_CPUSETS */

 #endif /* _LINUX_CPUSET_H */
diff -puN include/linux/mempolicy.h~task-cgroupsv11-make-cpusets-a-client-of-cgroups
include/linux/mempolicy.h
--- a/include/linux/mempolicy.h~task-cgroupsv11-make-cpusets-a-client-of-cgroups
+++ a/include/linux/mempolicy.h
@@ -147,14 +147,6 @@ extern void mpol_rebind_task(struct task
      const nodemask_t *new);
 extern void mpol_rebind_mm(struct mm_struct *mm, nodemask_t *new);
 extern void mpol_fix_fork_child_flag(struct task_struct *p);
-#define set_cpuset_being_rebound(x) (cpuset_being_rebound = (x))
-
-#ifdef CONFIG_CPUSETS
-#define current_cpuset_is_being_rebound() \
```

```
-    (cpuset_being_rebound == current->cpuset)
-#else
-#define current_cpuset_is_being_rebound() 0
-#endif

 extern struct mempolicy default_policy;
 extern struct zonelist *huge_zonelist(struct vm_area_struct *vma,
@@ -172,8 +164,6 @@ static inline void check_highest_zone(en
 int do_migrate_pages(struct mm_struct *mm,
  const nodemask_t *from_nodes, const nodemask_t *to_nodes, int flags);

-extern void *cpuset_being_rebound; /* Trigger mpol_copy vma rebind */
-
 #else

 struct mempolicy {};
@@ -247,8 +237,6 @@ static inline void mpol_fix_fork_child_f
 {
 }

-#define set_cpuset_being_rebound(x) do {} while (0)
-
 static inline struct zonelist *huge_zonelist(struct vm_area_struct *vma,
   unsigned long addr, gfp_t gfp_flags)
 {
diff -puN include/linux/sched.h~task-cgroupsv11-make-cpusets-a-client-of-cgroups
include/linux/sched.h
--- a/include/linux/sched.h~task-cgroupsv11-make-cpusets-a-client-of-cgroups
+++ a/include/linux/sched.h
@@ -730,8 +730,6 @@ static inline int above_background_load(
 }

 struct io_context;   /* See blkdev.h */
-struct cpuset;
-
 #define NGROUPS_SMALL  32
 #define NGROUPS_PER_BLOCK ((int)(PAGE_SIZE / sizeof(gid_t)))
 struct group_info {
@@ -1091,7 +1089,6 @@ struct task_struct {
  short il_next;
 #endif
 #ifdef CONFIG_CPUSETS
- struct cpuset *cpuset;
  nodemask_t mems_allowed;
  int cpuset_mems_generation;
  int cpuset_mem_spread_rotor;
diff -puN init/Kconfig~task-cgroupsv11-make-cpusets-a-client-of-cgroups init/Kconfig
--- a/init/Kconfig~task-cgroupsv11-make-cpusets-a-client-of-cgroups
```

```
+++ a/init/Kconfig
@@ -284,7 +284,7 @@ config CGROUPS

 config CPUSETS
  bool "Cpuset support"
- depends on SMP
+ depends on SMP && CGROUPS
  help
    This option will let you create and manage CPUSETs which
    allow dynamically partitioning a system into sets of CPUs and
@@ -313,6 +313,11 @@ config SYSFS_DEPRECATED
    If you are using a distro that was released in 2006 or later,
    it should be safe to say N here.

+config PROC_PID_CPUSET
+ bool "Include legacy /proc/<pid>/cpuset file"
+ depends on CPUSETS
+ default y
+
 config RELAY
  bool "Kernel->user space relay support (formerly relayfs)"
  help
diff -puN kernel/cpuset.c~task-cgroupsv11-make-cpusets-a-client-of-cgroups kernel/cpuset.c
--- a/kernel/cpuset.c~task-cgroupsv11-make-cpusets-a-client-of-cgroups
+++ a/kernel/cpuset.c
@@ -5,6 +5,7 @@
  *
  *  Copyright (C) 2003 BULL SA.
  *  Copyright (C) 2004-2006 Silicon Graphics, Inc.
+ *  Copyright (C) 2006 Google, Inc
  *
  *  Portions derived from Patrick Mochel's sysfs code.
  *  sysfs is Copyright (c) 2001-3 Patrick Mochel
@@ -12,6 +13,7 @@
  *  2003-10-10 Written by Simon Derr.
  *  2003-10-22 Updates by Stephen Hemminger.
  *  2004 May-July Rework by Paul Jackson.
+ *  2006 Rework by Paul Menage to use generic cgroups
  *
  *  This file is subject to the terms and conditions of the GNU General Public
  *  License.  See the file COPYING in the main directory of the Linux
@@ -53,8 +55,6 @@
 #include <asm/atomic.h>
 #include <linux/mutex.h>

-#define CPUSET_SUPER_MAGIC  0x27e0eb
-
 /*
```

```
  * Tracks how many cpusets are currently defined in system.
  * When there is only one cpuset (the root cpuset) we can
@@ -62,6 +62,10 @@
  */
 int number_of_cpusets __read_mostly;

+/* Retrieve the cpuset from a cgroup */
+struct cgroup_subsys cpuset_subsys;
+struct cpuset;
+
 /* See "Frequency meter" comments, below. */

 struct fmeter {
@@ -72,24 +76,13 @@ struct fmeter {
 };

 struct cpuset {
+ struct cgroup_subsys_state css;
+
  unsigned long flags;  /* "unsigned long" so bitops work */
  cpumask_t cpus_allowed;  /* CPUs allowed to tasks in cpuset */
  nodemask_t mems_allowed; /* Memory Nodes allowed to tasks */

- /*
-  * Count is atomic so can incr (fork) or decr (exit) without a lock.
-  */
- atomic_t count;   /* count tasks using this cpuset */
-
- /*
-  * We link our 'sibling' struct into our parents 'children'.
-  * Our children link their 'sibling' into our 'children'.
-  */
- struct list_head sibling; /* my parents children */
- struct list_head children; /* my children */
-
  struct cpuset *parent;  /* my parent */
- struct dentry *dentry;  /* cpuset fs entry */

  /*
   * Copy of global cpuset_mems_generation as of the most
@@ -100,13 +93,26 @@ struct cpuset {
  struct fmeter fmeter;  /* memory_pressure filter */
 };

+/* Retrieve the cpuset for a cgroup */
+static inline struct cpuset *cgroup_cs(struct cgroup *cont)
+{
+ return container_of(cgroup_subsys_state(cont, cpuset_subsys_id),
```

```
+       struct cpuset, css);
+}
+
+/* Retrieve the cpuset for a task */
+static inline struct cpuset *task_cs(struct task_struct *task)
+{
+ return container_of(task_subsys_state(task, cpuset_subsys_id),
+       struct cpuset, css);
+}
+
+
 /* bits in struct cpuset flags field */
 typedef enum {
   CS_CPU_EXCLUSIVE,
   CS_MEM_EXCLUSIVE,
   CS_MEMORY_MIGRATE,
- CS_REMOVED,
- CS_NOTIFY_ON_RELEASE,
   CS_SPREAD_PAGE,
   CS_SPREAD_SLAB,
 } cpuset_flagbits_t;
@@ -122,16 +128,6 @@ static inline int is_mem_exclusive(const
   return test_bit(CS_MEM_EXCLUSIVE, &cs->flags);
 }

-static inline int is_removed(const struct cpuset *cs)
-{
- return test_bit(CS_REMOVED, &cs->flags);
-}
-
-static inline int notify_on_release(const struct cpuset *cs)
-{
- return test_bit(CS_NOTIFY_ON_RELEASE, &cs->flags);
-}
-
 static inline int is_memory_migrate(const struct cpuset *cs)
 {
   return test_bit(CS_MEMORY_MIGRATE, &cs->flags);
@@ -172,14 +168,8 @@ static struct cpuset top_cpuset = {
   .flags = ((1 << CS_CPU_EXCLUSIVE) | (1 << CS_MEM_EXCLUSIVE)),
   .cpus_allowed = CPU_MASK_ALL,
   .mems_allowed = NODE_MASK_ALL,
- .count = ATOMIC_INIT(0),
- .sibling = LIST_HEAD_INIT(top_cpuset.sibling),
- .children = LIST_HEAD_INIT(top_cpuset.children),
 };

-static struct vfsmount *cpuset_mount;
```

```
-static struct super_block *cpuset_sb;
-
 /*
  * We have two global cpuset mutexes below.  They can nest.
  * It is ok to first take manage_mutex, then nest callback_mutex.  We also
@@ -263,297 +253,33 @@ static struct super_block *cpuset_sb;
  * the routine cpuset_update_task_memory_state().
  */

-static DEFINE_MUTEX(manage_mutex);
 static DEFINE_MUTEX(callback_mutex);

-/*
- * A couple of forward declarations required, due to cyclic reference loop:
- *  cpuset_mkdir -> cpuset_create -> cpuset_populate_dir -> cpuset_add_file
- *  -> cpuset_create_file -> cpuset_dir_inode_operations -> cpuset_mkdir.
- */
-
-static int cpuset_mkdir(struct inode *dir, struct dentry *dentry, int mode);
-static int cpuset_rmdir(struct inode *unused_dir, struct dentry *dentry);
-
-static struct backing_dev_info cpuset_backing_dev_info = {
- .ra_pages = 0,  /* No readahead */
- .capabilities = BDI_CAP_NO_ACCT_DIRTY | BDI_CAP_NO_WRITEBACK,
-};
-
-static struct inode *cpuset_new_inode(mode_t mode)
-{
- struct inode *inode = new_inode(cpuset_sb);
-
- if (inode) {
-  inode->i_mode = mode;
-  inode->i_uid = current->fsuid;
-  inode->i_gid = current->fsgid;
-  inode->i_blocks = 0;
-  inode->i_atime = inode->i_mtime = inode->i_ctime = CURRENT_TIME;
-  inode->i_mapping->backing_dev_info = &cpuset_backing_dev_info;
- }
- return inode;
-}
-
-static void cpuset_diput(struct dentry *dentry, struct inode *inode)
-{
- /* is dentry a directory ? if so, kfree() associated cpuset */
- if (S_ISDIR(inode->i_mode)) {
-  struct cpuset *cs = dentry->d_fsdata;
-  BUG_ON(!(is_removed(cs)));
-  kfree(cs);
```

```
- }
- iput(inode);
-}
-
-static struct dentry_operations cpuset_dops = {
- .d_iput = cpuset_diput,
-};
-
-static struct dentry *cpuset_get_dentry(struct dentry *parent, const char *name)
-{
- struct dentry *d = lookup_one_len(name, parent, strlen(name));
- if (!IS_ERR(d))
-  d->d_op = &cpuset_dops;
- return d;
-}
-
-static void remove_dir(struct dentry *d)
-{
- struct dentry *parent = dget(d->d_parent);
-
- d_delete(d);
- simple_rmdir(parent->d_inode, d);
- dput(parent);
-}
-
-/*
- * NOTE : the dentry must have been dget()'ed
- */
-static void cpuset_d_remove_dir(struct dentry *dentry)
-{
- struct list_head *node;
-
- spin_lock(&dcache_lock);
- node = dentry->d_subdirs.next;
- while (node != &dentry->d_subdirs) {
-  struct dentry *d = list_entry(node, struct dentry, d_u.d_child);
-  list_del_init(node);
-  if (d->d_inode) {
-   d = dget_locked(d);
-   spin_unlock(&dcache_lock);
-   d_delete(d);
-   simple_unlink(dentry->d_inode, d);
-   dput(d);
-   spin_lock(&dcache_lock);
-  }
-  node = dentry->d_subdirs.next;
- }
- list_del_init(&dentry->d_u.d_child);
```

```
- spin_unlock(&dcache_lock);
- remove_dir(dentry);
-}
-
-static struct super_operations cpuset_ops = {
- .statfs = simple_statfs,
- .drop_inode = generic_delete_inode,
-};
-
-static int cpuset_fill_super(struct super_block *sb, void *unused_data,
-       int unused_silent)
-{
- struct inode *inode;
- struct dentry *root;
-
- sb->s_blocksize = PAGE_CACHE_SIZE;
- sb->s_blocksize_bits = PAGE_CACHE_SHIFT;
- sb->s_magic = CPUSET_SUPER_MAGIC;
- sb->s_op = &cpuset_ops;
- cpuset_sb = sb;
-
- inode = cpuset_new_inode(S_IFDIR | S_IRUGO | S_IXUGO | S_IWUSR);
- if (inode) {
-  inode->i_op = &simple_dir_inode_operations;
-  inode->i_fop = &simple_dir_operations;
-  /* directories start off with i_nlink == 2 (for "." entry) */
-  inc_nlink(inode);
- } else {
-  return -ENOMEM;
- }
-
- root = d_alloc_root(inode);
- if (!root) {
-  iput(inode);
-  return -ENOMEM;
- }
- sb->s_root = root;
- return 0;
-}
-
+/* This is ugly, but preserves the userspace API for existing cpuset
+ * users. If someone tries to mount the "cpuset" filesystem, we
+ * silently switch it to mount "cgroup" instead */
 static int cpuset_get_sb(struct file_system_type *fs_type,
     int flags, const char *unused_dev_name,
     void *data, struct vfsmount *mnt)
 {
- return get_sb_single(fs_type, flags, data, cpuset_fill_super, mnt);
```

```
+ struct file_system_type *cgroup_fs = get_fs_type("cgroup");
+ int ret = -ENODEV;
+ if (cgroup_fs) {
+  char mountopts[] =
+   "cpuset,noprefix,"
+   "release_agent=/sbin/cpuset_release_agent";
+  ret = cgroup_fs->get_sb(cgroup_fs, flags,
+         unused_dev_name, mountopts, mnt);
+  put_filesystem(cgroup_fs);
+ }
+ return ret;
 }

 static struct file_system_type cpuset_fs_type = {
  .name = "cpuset",
  .get_sb = cpuset_get_sb,
- .kill_sb = kill_litter_super,
-};
-
-/* struct cftype:
- *
- * The files in the cpuset filesystem mostly have a very simple read/write
- * handling, some common function will take care of it. Nevertheless some cases
- * (read tasks) are special and therefore I define this structure for every
- * kind of file.
- *
- *
- * When reading/writing to a file:
- * - the cpuset to use in file->f_path.dentry->d_parent->d_fsdata
- * - the 'cftype' of the file is file->f_path.dentry->d_fsdata
- */
-
-struct cftype {
- char *name;
- int private;
- int (*open) (struct inode *inode, struct file *file);
- ssize_t (*read) (struct file *file, char __user *buf, size_t nbytes,
-       loff_t *ppos);
- int (*write) (struct file *file, const char __user *buf, size_t nbytes,
-       loff_t *ppos);
- int (*release) (struct inode *inode, struct file *file);
 };

-static inline struct cpuset *__d_cs(struct dentry *dentry)
-{
- return dentry->d_fsdata;
-}
-
```

```
-static inline struct cftype *__d_cft(struct dentry *dentry)
-{
- return dentry->d_fsdata;
-}
-
-/*
- * Call with manage_mutex held.  Writes path of cpuset into buf.
- * Returns 0 on success, -errno on error.
- */
-
-static int cpuset_path(const struct cpuset *cs, char *buf, int buflen)
-{
- char *start;
-
- start = buf + buflen;
-
- *--start = '\0';
- for (;;) {
-  int len = cs->dentry->d_name.len;
-  if ((start -= len) < buf)
-   return -ENAMETOOLONG;
-  memcpy(start, cs->dentry->d_name.name, len);
-  cs = cs->parent;
-  if (!cs)
-   break;
-  if (!cs->parent)
-   continue;
-  if (--start < buf)
-   return -ENAMETOOLONG;
-  *start = '/';
- }
- memmove(buf, start, buf + buflen - start);
- return 0;
-}
-
-/*
- * Notify userspace when a cpuset is released, by running
- * /sbin/cpuset_release_agent with the name of the cpuset (path
- * relative to the root of cpuset file system) as the argument.
- *
- * Most likely, this user command will try to rmdir this cpuset.
- *
- * This races with the possibility that some other task will be
- * attached to this cpuset before it is removed, or that some other
- * user task will 'mkdir' a child cpuset of this cpuset.  That's ok.
- * The presumed 'rmdir' will fail quietly if this cpuset is no longer
- * unused, and this cpuset will be reprieved from its death sentence,
- * to continue to serve a useful existence.  Next time it's released,
```

```
- * we will get notified again, if it still has 'notify_on_release' set.
- *
- * The final arg to call_usermodehelper() is 0, which means don't
- * wait.  The separate /sbin/cpuset_release_agent task is forked by
- * call_usermodehelper(), then control in this thread returns here,
- * without waiting for the release agent task.  We don't bother to
- * wait because the caller of this routine has no use for the exit
- * status of the /sbin/cpuset_release_agent task, so no sense holding
- * our caller up for that.
- *
- * When we had only one cpuset mutex, we had to call this
- * without holding it, to avoid deadlock when call_usermodehelper()
- * allocated memory.  With two locks, we could now call this while
- * holding manage_mutex, but we still don't, so as to minimize
- * the time manage_mutex is held.
- */
-
-static void cpuset_release_agent(const char *pathbuf)
-{
- char *argv[3], *envp[3];
- int i;
-
- if (!pathbuf)
-  return;
-
- i = 0;
- argv[i++] = "/sbin/cpuset_release_agent";
- argv[i++] = (char *)pathbuf;
- argv[i] = NULL;
-
- i = 0;
- /* minimal command environment */
- envp[i++] = "HOME=/";
- envp[i++] = "PATH=/sbin:/bin:/usr/sbin:/usr/bin";
- envp[i] = NULL;
-
- call_usermodehelper(argv[0], argv, envp, UMH_WAIT_EXEC);
- kfree(pathbuf);
-}
-
-/*
- * Either cs->count of using tasks transitioned to zero, or the
- * cs->children list of child cpusets just became empty.  If this
- * cs is notify_on_release() and now both the user count is zero and
- * the list of children is empty, prepare cpuset path in a kmalloc'd
- * buffer, to be returned via ppathbuf, so that the caller can invoke
- * cpuset_release_agent() with it later on, once manage_mutex is dropped.
- * Call here with manage_mutex held.
```

```
- *
- * This check_for_release() routine is responsible for kmalloc'ing
- * pathbuf.  The above cpuset_release_agent() is responsible for
- * kfree'ing pathbuf.  The caller of these routines is responsible
- * for providing a pathbuf pointer, initialized to NULL, then
- * calling check_for_release() with manage_mutex held and the address
- * of the pathbuf pointer, then dropping manage_mutex, then calling
- * cpuset_release_agent() with pathbuf, as set by check_for_release().
- */
-
-static void check_for_release(struct cpuset *cs, char **ppathbuf)
-{
- if (notify_on_release(cs) && atomic_read(&cs->count) == 0 &&
-     list_empty(&cs->children)) {
-  char *buf;
-
-  buf = kmalloc(PAGE_SIZE, GFP_KERNEL);
-  if (!buf)
-   return;
-  if (cpuset_path(cs, buf, PAGE_SIZE) < 0)
-   kfree(buf);
-  else
-   *ppathbuf = buf;
- }
-}
-
 /*
  * Return in *pmask the portion of a cpusets's cpus_allowed that
  * are online.  If none are online, walk up the cpuset hierarchy
@@ -653,20 +379,19 @@ void cpuset_update_task_memory_state(voi
  struct task_struct *tsk = current;
  struct cpuset *cs;

- if (tsk->cpuset == &top_cpuset) {
+ if (task_cs(tsk) == &top_cpuset) {
   /* Don't need rcu for top_cpuset.  It's never freed. */
   my_cpusets_mem_gen = top_cpuset.mems_generation;
  } else {
   rcu_read_lock();
-  cs = rcu_dereference(tsk->cpuset);
-  my_cpusets_mem_gen = cs->mems_generation;
+  my_cpusets_mem_gen = task_cs(current)->mems_generation;
   rcu_read_unlock();
  }

  if (my_cpusets_mem_gen != tsk->cpuset_mems_generation) {
   mutex_lock(&callback_mutex);
   task_lock(tsk);
```

```
-  cs = tsk->cpuset; /* Maybe changed when task not locked */
+  cs = task_cs(tsk); /* Maybe changed when task not locked */
   guarantee_online_mems(cs, &tsk->mems_allowed);
   tsk->cpuset_mems_generation = cs->mems_generation;
   if (is_spread_page(cs))
@@ -721,11 +446,12 @@ static int is_cpuset_subset(const struct

 static int validate_change(const struct cpuset *cur, const struct cpuset *trial)
 {
+ struct cgroup *cont;
  struct cpuset *c, *par;

  /* Each of our child cpusets must be a subset of us */
- list_for_each_entry(c, &cur->children, sibling) {
-  if (!is_cpuset_subset(c, trial))
+ list_for_each_entry(cont, &cur->css.cgroup->children, sibling) {
+  if (!is_cpuset_subset(cgroup_cs(cont), trial))
    return -EBUSY;
  }

@@ -740,7 +466,8 @@ static int validate_change(const struct
   return -EACCES;

  /* If either I or some sibling (!= me) is exclusive, we can't overlap */
- list_for_each_entry(c, &par->children, sibling) {
+ list_for_each_entry(cont, &par->css.cgroup->children, sibling) {
+  c = cgroup_cs(cont);
   if ((is_cpu_exclusive(trial) || is_cpu_exclusive(c)) &&
       c != cur &&
       cpus_intersects(trial->cpus_allowed, c->cpus_allowed))
@@ -783,7 +510,8 @@ static int update_cpumask(struct cpuset
  }
  cpus_and(trialcs.cpus_allowed, trialcs.cpus_allowed, cpu_online_map);
  /* cpus_allowed cannot be empty for a cpuset with attached tasks. */
- if (atomic_read(&cs->count) && cpus_empty(trialcs.cpus_allowed))
+ if (cgroup_task_count(cs->css.cgroup) &&
+     cpus_empty(trialcs.cpus_allowed))
   return -ENOSPC;
  retval = validate_change(cs, &trialcs);
  if (retval < 0)
@@ -839,7 +567,7 @@ static void cpuset_migrate_mm(struct mm_
  do_migrate_pages(mm, from, to, MPOL_MF_MOVE_ALL);

  mutex_lock(&callback_mutex);
- guarantee_online_mems(tsk->cpuset, &tsk->mems_allowed);
+ guarantee_online_mems(task_cs(tsk),&tsk->mems_allowed);
  mutex_unlock(&callback_mutex);
 }
```

```
@@ -857,16 +585,19 @@ static void cpuset_migrate_mm(struct mm_
  * their mempolicies to the cpusets new mems_allowed.
  */

+static void *cpuset_being_rebound;
+
 static int update_nodemask(struct cpuset *cs, char *buf)
 {
  struct cpuset trialcs;
  nodemask_t oldmem;
- struct task_struct *g, *p;
+ struct task_struct *p;
  struct mm_struct **mmarray;
  int i, n, ntasks;
  int migrate;
  int fudge;
  int retval;
+ struct cgroup_iter it;

  /*
   * top_cpuset.mems_allowed tracks node_stats[N_HIGH_MEMORY];
@@ -909,7 +640,8 @@ static int update_nodemask(struct cpuset
   goto done;
  }
  /* mems_allowed cannot be empty for a cpuset with attached tasks. */
- if (atomic_read(&cs->count) && nodes_empty(trialcs.mems_allowed)) {
+ if (cgroup_task_count(cs->css.cgroup) &&
+     nodes_empty(trialcs.mems_allowed)) {
   retval = -ENOSPC;
   goto done;
  }
@@ -922,7 +654,7 @@ static int update_nodemask(struct cpuset
  cs->mems_generation = cpuset_mems_generation++;
  mutex_unlock(&callback_mutex);

- set_cpuset_being_rebound(cs);  /* causes mpol_copy() rebind */
+ cpuset_being_rebound = cs;  /* causes mpol_copy() rebind */

  fudge = 10;    /* spare mmarray[] slots */
  fudge += cpus_weight(cs->cpus_allowed); /* imagine one fork-bomb/cpu */
@@ -936,13 +668,13 @@ static int update_nodemask(struct cpuset
   * enough mmarray[] w/o using GFP_ATOMIC.
   */
  while (1) {
-  ntasks = atomic_read(&cs->count); /* guess */
+  ntasks = cgroup_task_count(cs->css.cgroup);  /* guess */
   ntasks += fudge;
```

```
   mmarray = kmalloc(ntasks * sizeof(*mmarray), GFP_KERNEL);
   if (!mmarray)
    goto done;
   read_lock(&tasklist_lock);  /* block fork */
-  if (atomic_read(&cs->count) <= ntasks)
+  if (cgroup_task_count(cs->css.cgroup) <= ntasks)
    break;    /* got enough */
   read_unlock(&tasklist_lock);  /* try again */
   kfree(mmarray);
@@ -951,21 +683,21 @@ static int update_nodemask(struct cpuset
  n = 0;

  /* Load up mmarray[] with mm reference for each task in cpuset. */
- do_each_thread(g, p) {
+ cgroup_iter_start(cs->css.cgroup, &it);
+ while ((p = cgroup_iter_next(cs->css.cgroup, &it))) {
   struct mm_struct *mm;

   if (n >= ntasks) {
    printk(KERN_WARNING
     "Cpuset mempolicy rebind incomplete.\n");
-   continue;
+   break;
   }
-  if (p->cpuset != cs)
-   continue;
   mm = get_task_mm(p);
   if (!mm)
    continue;
   mmarray[n++] = mm;
- } while_each_thread(g, p);
+ }
+ cgroup_iter_end(cs->css.cgroup, &it);
  read_unlock(&tasklist_lock);

  /*
@@ -993,12 +725,17 @@ static int update_nodemask(struct cpuset

  /* We're done rebinding vma's to this cpusets new mems_allowed. */
  kfree(mmarray);
- set_cpuset_being_rebound(NULL);
+ cpuset_being_rebound = NULL;
  retval = 0;
 done:
  return retval;
 }

+int current_cpuset_is_being_rebound(void)
```

```
+{
+ return task_cs(current) == cpuset_being_rebound;
+}
+
 /*
  * Call with manage_mutex held.
  */
@@ -1145,85 +882,34 @@ static int fmeter_getrate(struct fmeter
  return val;
 }

-/*
- * Attack task specified by pid in 'pidbuf' to cpuset 'cs', possibly
- * writing the path of the old cpuset in 'ppathbuf' if it needs to be
- * notified on release.
- *
- * Call holding manage_mutex.  May take callback_mutex and task_lock of
- * the task 'pid' during call.
- */
-
-static int attach_task(struct cpuset *cs, char *pidbuf, char **ppathbuf)
+static int cpuset_can_attach(struct cgroup_subsys *ss,
+        struct cgroup *cont, struct task_struct *tsk)
 {
- pid_t pid;
- struct task_struct *tsk;
- struct cpuset *oldcs;
- cpumask_t cpus;
- nodemask_t from, to;
- struct mm_struct *mm;
- int retval;
+ struct cpuset *cs = cgroup_cs(cont);

- if (sscanf(pidbuf, "%d", &pid) != 1)
-  return -EIO;
  if (cpus_empty(cs->cpus_allowed) || nodes_empty(cs->mems_allowed))
   return -ENOSPC;

- if (pid) {
-  read_lock(&tasklist_lock);
-
-  tsk = find_task_by_pid(pid);
-  if (!tsk || tsk->flags & PF_EXITING) {
-   read_unlock(&tasklist_lock);
-   return -ESRCH;
-  }
-
-  get_task_struct(tsk);
```

```
-  read_unlock(&tasklist_lock);
-
-  if ((current->euid) && (current->euid != tsk->uid)
-      && (current->euid != tsk->suid)) {
-   put_task_struct(tsk);
-   return -EACCES;
-  }
- } else {
-  tsk = current;
-  get_task_struct(tsk);
- }
+ return security_task_setscheduler(tsk, 0, NULL);
+}

- retval = security_task_setscheduler(tsk, 0, NULL);
- if (retval) {
-  put_task_struct(tsk);
-  return retval;
- }
+static void cpuset_attach(struct cgroup_subsys *ss,
+      struct cgroup *cont, struct cgroup *oldcont,
+      struct task_struct *tsk)
+{
+ cpumask_t cpus;
+ nodemask_t from, to;
+ struct mm_struct *mm;
+ struct cpuset *cs = cgroup_cs(cont);
+ struct cpuset *oldcs = cgroup_cs(oldcont);

  mutex_lock(&callback_mutex);
-
- task_lock(tsk);
- oldcs = tsk->cpuset;
- /*
-  * After getting 'oldcs' cpuset ptr, be sure still not exiting.
-  * If 'oldcs' might be the top_cpuset due to the_top_cpuset_hack
-  * then fail this attach_task(), to avoid breaking top_cpuset.count.
-  */
- if (tsk->flags & PF_EXITING) {
-  task_unlock(tsk);
-  mutex_unlock(&callback_mutex);
-  put_task_struct(tsk);
-  return -ESRCH;
- }
- atomic_inc(&cs->count);
- rcu_assign_pointer(tsk->cpuset, cs);
- task_unlock(tsk);
-
```

```
  guarantee_online_cpus(cs, &cpus);
  set_cpus_allowed(tsk, cpus);
+ mutex_unlock(&callback_mutex);

  from = oldcs->mems_allowed;
  to = cs->mems_allowed;
-
- mutex_unlock(&callback_mutex);
-
  mm = get_task_mm(tsk);
  if (mm) {
   mpol_rebind_mm(mm, &to);
@@ -1232,40 +918,31 @@ static int attach_task(struct cpuset *cs
   mmput(mm);
  }

- put_task_struct(tsk);
- synchronize_rcu();
- if (atomic_dec_and_test(&oldcs->count))
-  check_for_release(oldcs, ppathbuf);
- return 0;
 }

 /* The various types of files and directories in a cpuset file system */

 typedef enum {
- FILE_ROOT,
- FILE_DIR,
  FILE_MEMORY_MIGRATE,
  FILE_CPULIST,
  FILE_MEMLIST,
  FILE_CPU_EXCLUSIVE,
  FILE_MEM_EXCLUSIVE,
- FILE_NOTIFY_ON_RELEASE,
  FILE_MEMORY_PRESSURE_ENABLED,
  FILE_MEMORY_PRESSURE,
  FILE_SPREAD_PAGE,
  FILE_SPREAD_SLAB,
- FILE_TASKLIST,
 } cpuset_filetype_t;

-static ssize_t cpuset_common_file_write(struct file *file,
+static ssize_t cpuset_common_file_write(struct cgroup *cont,
+      struct cftype *cft,
+      struct file *file,
       const char __user *userbuf,
       size_t nbytes, loff_t *unused_ppos)
 {
```

```
- struct cpuset *cs = __d_cs(file->f_path.dentry->d_parent);
- struct cftype *cft = __d_cft(file->f_path.dentry);
+ struct cpuset *cs = cgroup_cs(cont);
  cpuset_filetype_t type = cft->private;
  char *buffer;
- char *pathbuf = NULL;
  int retval = 0;

  /* Crude upper limit on largest legitimate cpulist user might write. */
@@ -1282,9 +959,9 @@ static ssize_t cpuset_common_file_write(
  }
  buffer[nbytes] = 0; /* nul-terminate */

- mutex_lock(&manage_mutex);
+ cgroup_lock();

- if (is_removed(cs)) {
+ if (cgroup_is_removed(cont)) {
   retval = -ENODEV;
   goto out2;
  }
@@ -1302,9 +979,6 @@ static ssize_t cpuset_common_file_write(
  case FILE_MEM_EXCLUSIVE:
   retval = update_flag(CS_MEM_EXCLUSIVE, cs, buffer);
   break;
- case FILE_NOTIFY_ON_RELEASE:
-  retval = update_flag(CS_NOTIFY_ON_RELEASE, cs, buffer);
-  break;
  case FILE_MEMORY_MIGRATE:
   retval = update_flag(CS_MEMORY_MIGRATE, cs, buffer);
   break;
@@ -1322,9 +996,6 @@ static ssize_t cpuset_common_file_write(
   retval = update_flag(CS_SPREAD_SLAB, cs, buffer);
   cs->mems_generation = cpuset_mems_generation++;
   break;
- case FILE_TASKLIST:
-  retval = attach_task(cs, buffer, &pathbuf);
-  break;
  default:
   retval = -EINVAL;
   goto out2;
@@ -1333,30 +1004,12 @@ static ssize_t cpuset_common_file_write(
  if (retval == 0)
   retval = nbytes;
 out2:
- mutex_unlock(&manage_mutex);
- cpuset_release_agent(pathbuf);
+ cgroup_unlock();
```

```
 out1:
  kfree(buffer);
  return retval;
 }

-static ssize_t cpuset_file_write(struct file *file, const char __user *buf,
-      size_t nbytes, loff_t *ppos)
-{
- ssize_t retval = 0;
- struct cftype *cft = __d_cft(file->f_path.dentry);
- if (!cft)
-  return -ENODEV;
-
- /* special function ? */
- if (cft->write)
-  retval = cft->write(file, buf, nbytes, ppos);
- else
-  retval = cpuset_common_file_write(file, buf, nbytes, ppos);
-
- return retval;
-}
-
 /*
  * These ascii lists should be read in a single call, by using a user
  * buffer large enough to hold the entire map.  If read in smaller
@@ -1391,11 +1044,13 @@ static int cpuset_sprintf_memlist(char *
  return nodelist_scnprintf(page, PAGE_SIZE, mask);
 }

-static ssize_t cpuset_common_file_read(struct file *file, char __user *buf,
-    size_t nbytes, loff_t *ppos)
+static ssize_t cpuset_common_file_read(struct cgroup *cont,
+           struct cftype *cft,
+           struct file *file,
+           char __user *buf,
+           size_t nbytes, loff_t *ppos)
 {
- struct cftype *cft = __d_cft(file->f_path.dentry);
- struct cpuset *cs = __d_cs(file->f_path.dentry->d_parent);
+ struct cpuset *cs = cgroup_cs(cont);
  cpuset_filetype_t type = cft->private;
  char *page;
  ssize_t retval = 0;
@@ -1419,9 +1074,6 @@ static ssize_t cpuset_common_file_read(s
  case FILE_MEM_EXCLUSIVE:
   *s++ = is_mem_exclusive(cs) ? '1' : '0';
   break;
- case FILE_NOTIFY_ON_RELEASE:
```

```
-  *s++ = notify_on_release(cs) ? '1' : '0';
-  break;
   case FILE_MEMORY_MIGRATE:
    *s++ = is_memory_migrate(cs) ? '1' : '0';
    break;
@@ -1449,390 +1101,141 @@ out:
  return retval;
 }

-static ssize_t cpuset_file_read(struct file *file, char __user *buf, size_t nbytes,
-        loff_t *ppos)
-{
- ssize_t retval = 0;
- struct cftype *cft = __d_cft(file->f_path.dentry);
- if (!cft)
-  return -ENODEV;
-
- /* special function ? */
- if (cft->read)
-  retval = cft->read(file, buf, nbytes, ppos);
- else
-  retval = cpuset_common_file_read(file, buf, nbytes, ppos);
-
- return retval;
-}
-
-static int cpuset_file_open(struct inode *inode, struct file *file)
-{
- int err;
- struct cftype *cft;
-
- err = generic_file_open(inode, file);
- if (err)
-  return err;
-
- cft = __d_cft(file->f_path.dentry);
- if (!cft)
-  return -ENODEV;
- if (cft->open)
-  err = cft->open(inode, file);
- else
-  err = 0;
-
- return err;
-}
-
-static int cpuset_file_release(struct inode *inode, struct file *file)
-{
```

```
- struct cftype *cft = __d_cft(file->f_path.dentry);
- if (cft->release)
-  return cft->release(inode, file);
- return 0;
-}
-
-/*
- * cpuset_rename - Only allow simple rename of directories in place.
- */
-static int cpuset_rename(struct inode *old_dir, struct dentry *old_dentry,
-                         struct inode *new_dir, struct dentry *new_dentry)
-{
- if (!S_ISDIR(old_dentry->d_inode->i_mode))
-  return -ENOTDIR;
- if (new_dentry->d_inode)
-  return -EEXIST;
- if (old_dir != new_dir)
-  return -EIO;
- return simple_rename(old_dir, old_dentry, new_dir, new_dentry);
-}
-
-static const struct file_operations cpuset_file_operations = {
- .read = cpuset_file_read,
- .write = cpuset_file_write,
- .llseek = generic_file_llseek,
- .open = cpuset_file_open,
- .release = cpuset_file_release,
-};
-
-static const struct inode_operations cpuset_dir_inode_operations = {
- .lookup = simple_lookup,
- .mkdir = cpuset_mkdir,
- .rmdir = cpuset_rmdir,
- .rename = cpuset_rename,
-};
-
-static int cpuset_create_file(struct dentry *dentry, int mode)
-{
- struct inode *inode;
-
- if (!dentry)
-  return -ENOENT;
- if (dentry->d_inode)
-  return -EEXIST;

- inode = cpuset_new_inode(mode);
- if (!inode)
-  return -ENOMEM;
```

```
-
- if (S_ISDIR(mode)) {
-  inode->i_op = &cpuset_dir_inode_operations;
-  inode->i_fop = &simple_dir_operations;
-
-  /* start off with i_nlink == 2 (for "." entry) */
-  inc_nlink(inode);
- } else if (S_ISREG(mode)) {
-  inode->i_size = 0;
-  inode->i_fop = &cpuset_file_operations;
- }
-
- d_instantiate(dentry, inode);
- dget(dentry); /* Extra count - pin the dentry in core */
- return 0;
-}
-
-/*
- * cpuset_create_dir - create a directory for an object.
- * cs: the cpuset we create the directory for.
- *  It must have a valid ->parent field
- *  And we are going to fill its ->dentry field.
- * name: The name to give to the cpuset directory. Will be copied.
- * mode: mode to set on new directory.
- */
-
-static int cpuset_create_dir(struct cpuset *cs, const char *name, int mode)
-{
- struct dentry *dentry = NULL;
- struct dentry *parent;
- int error = 0;

- parent = cs->parent->dentry;
- dentry = cpuset_get_dentry(parent, name);
- if (IS_ERR(dentry))
-  return PTR_ERR(dentry);
- error = cpuset_create_file(dentry, S_IFDIR | mode);
- if (!error) {
-  dentry->d_fsdata = cs;
-  inc_nlink(parent->d_inode);
-  cs->dentry = dentry;
- }
- dput(dentry);
-
- return error;
-}

-static int cpuset_add_file(struct dentry *dir, const struct cftype *cft)
```

```
-{
- struct dentry *dentry;
- int error;
-
- mutex_lock(&dir->d_inode->i_mutex);
- dentry = cpuset_get_dentry(dir, cft->name);
- if (!IS_ERR(dentry)) {
-  error = cpuset_create_file(dentry, 0644 | S_IFREG);
-  if (!error)
-   dentry->d_fsdata = (void *)cft;
-  dput(dentry);
- } else
-  error = PTR_ERR(dentry);
- mutex_unlock(&dir->d_inode->i_mutex);
- return error;
-}
-
-/*
- * Stuff for reading the 'tasks' file.
- *
- * Reading this file can return large amounts of data if a cpuset has
- * *lots* of attached tasks. So it may need several calls to read(),
- * but we cannot guarantee that the information we produce is correct
- * unless we produce it entirely atomically.
- *
- * Upon tasks file open(), a struct ctr_struct is allocated, that
- * will have a pointer to an array (also allocated here).  The struct
- * ctr_struct * is stored in file->private_data.  Its resources will
- * be freed by release() when the file is closed.  The array is used
- * to sprintf the PIDs and then used by read().
- */
-
-/* cpusets_tasks_read array */
-
-struct ctr_struct {
- char *buf;
- int bufsz;
-};
-
-/*
- * Load into 'pidarray' up to 'npids' of the tasks using cpuset 'cs'.
- * Return actual number of pids loaded.  No need to task_lock(p)
- * when reading out p->cpuset, as we don't really care if it changes
- * on the next cycle, and we are not going to try to dereference it.
- */
-static int pid_array_load(pid_t *pidarray, int npids, struct cpuset *cs)
-{
- int n = 0;
```

```
- struct task_struct *g, *p;
-
- read_lock(&tasklist_lock);
-
- do_each_thread(g, p) {
-  if (p->cpuset == cs) {
-   pidarray[n++] = p->pid;
-   if (unlikely(n == npids))
-    goto array_full;
-  }
- } while_each_thread(g, p);
-
-array_full:
- read_unlock(&tasklist_lock);
- return n;
-}
-
-static int cmppid(const void *a, const void *b)
-{
- return *(pid_t *)a - *(pid_t *)b;
-}
-
-/*
- * Convert array 'a' of 'npids' pid_t's to a string of newline separated
- * decimal pids in 'buf'.  Don't write more than 'sz' chars, but return
- * count 'cnt' of how many chars would be written if buf were large enough.
- */
-static int pid_array_to_buf(char *buf, int sz, pid_t *a, int npids)
-{
- int cnt = 0;
- int i;
-
- for (i = 0; i < npids; i++)
-  cnt += snprintf(buf + cnt, max(sz - cnt, 0), "%d\n", a[i]);
- return cnt;
-}
-
-/*
- * Handle an open on 'tasks' file.  Prepare a buffer listing the
- * process id's of tasks currently attached to the cpuset being opened.
- *
- * Does not require any specific cpuset mutexes, and does not take any.
- */
-static int cpuset_tasks_open(struct inode *unused, struct file *file)
-{
- struct cpuset *cs = __d_cs(file->f_path.dentry->d_parent);
- struct ctr_struct *ctr;
- pid_t *pidarray;
```

```
- int npids;
- char c;
-
- if (!(file->f_mode & FMODE_READ))
-  return 0;
-
- ctr = kmalloc(sizeof(*ctr), GFP_KERNEL);
- if (!ctr)
-  goto err0;
-
- /*
-  * If cpuset gets more users after we read count, we won't have
-  * enough space - tough.  This race is indistinguishable to the
-  * caller from the case that the additional cpuset users didn't
-  * show up until sometime later on.
-  */
- npids = atomic_read(&cs->count);
- pidarray = kmalloc(npids * sizeof(pid_t), GFP_KERNEL);
- if (!pidarray)
-  goto err1;
-
- npids = pid_array_load(pidarray, npids, cs);
- sort(pidarray, npids, sizeof(pid_t), cmppid, NULL);
-
- /* Call pid_array_to_buf() twice, first just to get bufsz */
- ctr->bufsz = pid_array_to_buf(&c, sizeof(c), pidarray, npids) + 1;
- ctr->buf = kmalloc(ctr->bufsz, GFP_KERNEL);
- if (!ctr->buf)
-  goto err2;
- ctr->bufsz = pid_array_to_buf(ctr->buf, ctr->bufsz, pidarray, npids);
-
- kfree(pidarray);
- file->private_data = ctr;
- return 0;
-
-err2:
- kfree(pidarray);
-err1:
- kfree(ctr);
-err0:
- return -ENOMEM;
-}
-
-static ssize_t cpuset_tasks_read(struct file *file, char __user *buf,
-      size_t nbytes, loff_t *ppos)
-{
- struct ctr_struct *ctr = file->private_data;
-
```

```
- return simple_read_from_buffer(buf, nbytes, ppos, ctr->buf, ctr->bufsz);
-}
-
-static int cpuset_tasks_release(struct inode *unused_inode, struct file *file)
-{
- struct ctr_struct *ctr;
-
- if (file->f_mode & FMODE_READ) {
-  ctr = file->private_data;
-  kfree(ctr->buf);
-  kfree(ctr);
- }
- return 0;
-}

 /*
  * for the common functions, 'private' gives the type of file
  */

-static struct cftype cft_tasks = {
- .name = "tasks",
- .open = cpuset_tasks_open,
- .read = cpuset_tasks_read,
- .release = cpuset_tasks_release,
- .private = FILE_TASKLIST,
-};
-
 static struct cftype cft_cpus = {
  .name = "cpus",
+ .read = cpuset_common_file_read,
+ .write = cpuset_common_file_write,
  .private = FILE_CPULIST,
 };

 static struct cftype cft_mems = {
  .name = "mems",
+ .read = cpuset_common_file_read,
+ .write = cpuset_common_file_write,
  .private = FILE_MEMLIST,
 };

 static struct cftype cft_cpu_exclusive = {
  .name = "cpu_exclusive",
+ .read = cpuset_common_file_read,
+ .write = cpuset_common_file_write,
  .private = FILE_CPU_EXCLUSIVE,
 };
```

```
 static struct cftype cft_mem_exclusive = {
  .name = "mem_exclusive",
+ .read = cpuset_common_file_read,
+ .write = cpuset_common_file_write,
  .private = FILE_MEM_EXCLUSIVE,
 };

-static struct cftype cft_notify_on_release = {
- .name = "notify_on_release",
- .private = FILE_NOTIFY_ON_RELEASE,
-};
-
 static struct cftype cft_memory_migrate = {
  .name = "memory_migrate",
+ .read = cpuset_common_file_read,
+ .write = cpuset_common_file_write,
  .private = FILE_MEMORY_MIGRATE,
 };

 static struct cftype cft_memory_pressure_enabled = {
  .name = "memory_pressure_enabled",
+ .read = cpuset_common_file_read,
+ .write = cpuset_common_file_write,
  .private = FILE_MEMORY_PRESSURE_ENABLED,
 };

 static struct cftype cft_memory_pressure = {
  .name = "memory_pressure",
+ .read = cpuset_common_file_read,
+ .write = cpuset_common_file_write,
  .private = FILE_MEMORY_PRESSURE,
 };

 static struct cftype cft_spread_page = {
  .name = "memory_spread_page",
+ .read = cpuset_common_file_read,
+ .write = cpuset_common_file_write,
  .private = FILE_SPREAD_PAGE,
 };

 static struct cftype cft_spread_slab = {
  .name = "memory_spread_slab",
+ .read = cpuset_common_file_read,
+ .write = cpuset_common_file_write,
  .private = FILE_SPREAD_SLAB,
 };

-static int cpuset_populate_dir(struct dentry *cs_dentry)
```

```
+static int cpuset_populate(struct cgroup_subsys *ss, struct cgroup *cont)
 {
  int err;

- if ((err = cpuset_add_file(cs_dentry, &cft_cpus)) < 0)
+ if ((err = cgroup_add_file(cont, ss, &cft_cpus)) < 0)
    return err;
- if ((err = cpuset_add_file(cs_dentry, &cft_mems)) < 0)
+ if ((err = cgroup_add_file(cont, ss, &cft_mems)) < 0)
    return err;
- if ((err = cpuset_add_file(cs_dentry, &cft_cpu_exclusive)) < 0)
+ if ((err = cgroup_add_file(cont, ss, &cft_cpu_exclusive)) < 0)
    return err;
- if ((err = cpuset_add_file(cs_dentry, &cft_mem_exclusive)) < 0)
+ if ((err = cgroup_add_file(cont, ss, &cft_mem_exclusive)) < 0)
    return err;
- if ((err = cpuset_add_file(cs_dentry, &cft_notify_on_release)) < 0)
+ if ((err = cgroup_add_file(cont, ss, &cft_memory_migrate)) < 0)
    return err;
- if ((err = cpuset_add_file(cs_dentry, &cft_memory_migrate)) < 0)
+ if ((err = cgroup_add_file(cont, ss, &cft_memory_pressure)) < 0)
    return err;
- if ((err = cpuset_add_file(cs_dentry, &cft_memory_pressure)) < 0)
+ if ((err = cgroup_add_file(cont, ss, &cft_spread_page)) < 0)
    return err;
- if ((err = cpuset_add_file(cs_dentry, &cft_spread_page)) < 0)
-   return err;
- if ((err = cpuset_add_file(cs_dentry, &cft_spread_slab)) < 0)
-   return err;
- if ((err = cpuset_add_file(cs_dentry, &cft_tasks)) < 0)
+ if ((err = cgroup_add_file(cont, ss, &cft_spread_slab)) < 0)
    return err;
+ /* memory_pressure_enabled is in root cpuset only */
+ if (err == 0 && !cont->parent)
+   err = cgroup_add_file(cont, ss,
+         &cft_memory_pressure_enabled);
  return 0;
 }

 /*
+ * post_clone() is called at the end of cgroup_clone().
+ * 'cgroup' was just created automatically as a result of
+ * a cgroup_clone(), and the current task is about to
+ * be moved into 'cgroup'.
+ *
+ * Currently we refuse to set up the cgroup - thereby
+ * refusing the task to be entered, and as a result refusing
+ * the sys_unshare() or clone() which initiated it - if any
```

```
+ * sibling cpusets have exclusive cpus or mem.
+ *
+ * If this becomes a problem for some users who wish to
+ * allow that scenario, then cpuset_post_clone() could be
+ * changed to grant parent->cpus_allowed-sibling_cpus_exclusive
+ * (and likewise for mems) to the new cgroup.
+ */
+static void cpuset_post_clone(struct cgroup_subsys *ss,
+         struct cgroup *cgroup)
+{
+ struct cgroup *parent, *child;
+ struct cpuset *cs, *parent_cs;
+
+ parent = cgroup->parent;
+ list_for_each_entry(child, &parent->children, sibling) {
+  cs = cgroup_cs(child);
+  if (is_mem_exclusive(cs) || is_cpu_exclusive(cs))
+   return;
+ }
+ cs = cgroup_cs(cgroup);
+ parent_cs = cgroup_cs(parent);
+
+ cs->mems_allowed = parent_cs->mems_allowed;
+ cs->cpus_allowed = parent_cs->cpus_allowed;
+ return;
+}
+
+/*
  * cpuset_create - create a cpuset
  * parent: cpuset that will be parent of the new cpuset.
  * name:  name of the new cpuset. Will be strcpy'ed.
@@ -1841,106 +1244,60 @@ static int cpuset_populate_dir(struct de
  * Must be called with the mutex on the parent inode held
  */

-static long cpuset_create(struct cpuset *parent, const char *name, int mode)
+static struct cgroup_subsys_state *cpuset_create(
+ struct cgroup_subsys *ss,
+ struct cgroup *cont)
 {
  struct cpuset *cs;
- int err;
+ struct cpuset *parent;

+ if (!cont->parent) {
+  /* This is early initialization for the top cgroup */
+  top_cpuset.mems_generation = cpuset_mems_generation++;
+  return &top_cpuset.css;
```

```
+ }
+ parent = cgroup_cs(cont->parent);
  cs = kmalloc(sizeof(*cs), GFP_KERNEL);
  if (!cs)
-  return -ENOMEM;
+  return ERR_PTR(-ENOMEM);

- mutex_lock(&manage_mutex);
  cpuset_update_task_memory_state();
  cs->flags = 0;
- if (notify_on_release(parent))
-  set_bit(CS_NOTIFY_ON_RELEASE, &cs->flags);
  if (is_spread_page(parent))
   set_bit(CS_SPREAD_PAGE, &cs->flags);
  if (is_spread_slab(parent))
   set_bit(CS_SPREAD_SLAB, &cs->flags);
  cs->cpus_allowed = CPU_MASK_NONE;
  cs->mems_allowed = NODE_MASK_NONE;
- atomic_set(&cs->count, 0);
- INIT_LIST_HEAD(&cs->sibling);
- INIT_LIST_HEAD(&cs->children);
  cs->mems_generation = cpuset_mems_generation++;
  fmeter_init(&cs->fmeter);

  cs->parent = parent;
-
- mutex_lock(&callback_mutex);
- list_add(&cs->sibling, &cs->parent->children);
  number_of_cpusets++;
- mutex_unlock(&callback_mutex);
-
- err = cpuset_create_dir(cs, name, mode);
- if (err < 0)
-  goto err;
-
- /*
-  * Release manage_mutex before cpuset_populate_dir() because it
-  * will down() this new directory's i_mutex and if we race with
-  * another mkdir, we might deadlock.
-  */
- mutex_unlock(&manage_mutex);
-
- err = cpuset_populate_dir(cs->dentry);
- /* If err < 0, we have a half-filled directory - oh well ;) */
- return 0;
-err:
- list_del(&cs->sibling);
- mutex_unlock(&manage_mutex);
```

```
- kfree(cs);
- return err;
-}
-
-static int cpuset_mkdir(struct inode *dir, struct dentry *dentry, int mode)
-{
- struct cpuset *c_parent = dentry->d_parent->d_fsdata;
-
- /* the vfs holds inode->i_mutex already */
- return cpuset_create(c_parent, dentry->d_name.name, mode | S_IFDIR);
+ return &cs->css ;
 }

-static int cpuset_rmdir(struct inode *unused_dir, struct dentry *dentry)
+static void cpuset_destroy(struct cgroup_subsys *ss, struct cgroup *cont)
 {
- struct cpuset *cs = dentry->d_fsdata;
- struct dentry *d;
- struct cpuset *parent;
- char *pathbuf = NULL;
-
- /* the vfs holds both inode->i_mutex already */
+ struct cpuset *cs = cgroup_cs(cont);

- mutex_lock(&manage_mutex);
  cpuset_update_task_memory_state();
- if (atomic_read(&cs->count) > 0) {
-  mutex_unlock(&manage_mutex);
-  return -EBUSY;
- }
- if (!list_empty(&cs->children)) {
-  mutex_unlock(&manage_mutex);
-  return -EBUSY;
- }
- parent = cs->parent;
- mutex_lock(&callback_mutex);
- set_bit(CS_REMOVED, &cs->flags);
- list_del(&cs->sibling); /* delete my sibling from parent->children */
- spin_lock(&cs->dentry->d_lock);
- d = dget(cs->dentry);
- cs->dentry = NULL;
- spin_unlock(&d->d_lock);
- cpuset_d_remove_dir(d);
- dput(d);
  number_of_cpusets--;
- mutex_unlock(&callback_mutex);
- if (list_empty(&parent->children))
-  check_for_release(parent, &pathbuf);
```

```
- mutex_unlock(&manage_mutex);
- cpuset_release_agent(pathbuf);
- return 0;
+ kfree(cs);
 }

+struct cgroup_subsys cpuset_subsys = {
+ .name = "cpuset",
+ .create = cpuset_create,
+ .destroy  = cpuset_destroy,
+ .can_attach = cpuset_can_attach,
+ .attach = cpuset_attach,
+ .populate = cpuset_populate,
+ .post_clone = cpuset_post_clone,
+ .subsys_id = cpuset_subsys_id,
+ .early_init = 1,
+};
+
 /*
  * cpuset_init_early - just enough so that the calls to
  * cpuset_update_task_memory_state() in early init code
@@ -1949,13 +1306,11 @@ static int cpuset_rmdir(struct inode *un

 int __init cpuset_init_early(void)
 {
- struct task_struct *tsk = current;
-
- tsk->cpuset = &top_cpuset;
- tsk->cpuset->mems_generation = cpuset_mems_generation++;
+ top_cpuset.mems_generation = cpuset_mems_generation++;
  return 0;
 }

+
 /**
  * cpuset_init - initialize cpusets at system boot
  *
@@ -1964,8 +1319,7 @@ int __init cpuset_init_early(void)

 int __init cpuset_init(void)
 {
- struct dentry *root;
- int err;
+ int err = 0;

  top_cpuset.cpus_allowed = CPU_MASK_ALL;
  top_cpuset.mems_allowed = NODE_MASK_ALL;
@@ -1973,30 +1327,12 @@ int __init cpuset_init(void)
```

```
  fmeter_init(&top_cpuset.fmeter);
  top_cpuset.mems_generation = cpuset_mems_generation++;

- init_task.cpuset = &top_cpuset;
-
  err = register_filesystem(&cpuset_fs_type);
  if (err < 0)
-  goto out;
- cpuset_mount = kern_mount(&cpuset_fs_type);
- if (IS_ERR(cpuset_mount)) {
-  printk(KERN_ERR "cpuset: could not mount!\n");
-  err = PTR_ERR(cpuset_mount);
-  cpuset_mount = NULL;
-  goto out;
- }
- root = cpuset_mount->mnt_sb->s_root;
- root->d_fsdata = &top_cpuset;
- inc_nlink(root->d_inode);
- top_cpuset.dentry = root;
- root->d_inode->i_op = &cpuset_dir_inode_operations;
+  return err;
+
  number_of_cpusets = 1;
- err = cpuset_populate_dir(root);
- /* memory_pressure_enabled is in root cpuset only */
- if (err == 0)
-  err = cpuset_add_file(root, &cft_memory_pressure_enabled);
-out:
- return err;
+ return 0;
 }

 /*
@@ -2022,10 +1358,12 @@ out:

 static void guarantee_online_cpus_mems_in_subtree(const struct cpuset *cur)
 {
+ struct cgroup *cont;
  struct cpuset *c;

  /* Each of our child cpusets mems must be online */
- list_for_each_entry(c, &cur->children, sibling) {
+ list_for_each_entry(cont, &cur->css.cgroup->children, sibling) {
+  c = cgroup_cs(cont);
   guarantee_online_cpus_mems_in_subtree(c);
   if (!cpus_empty(c->cpus_allowed))
    guarantee_online_cpus(c, &c->cpus_allowed);
@@ -2053,7 +1391,7 @@ static void guarantee_online_cpus_mems_i
```

```
 static void common_cpu_mem_hotplug_unplug(void)
 {
- mutex_lock(&manage_mutex);
+ cgroup_lock();
  mutex_lock(&callback_mutex);

  guarantee_online_cpus_mems_in_subtree(&top_cpuset);
@@ -2061,7 +1399,7 @@ static void common_cpu_mem_hotplug_unplu
  top_cpuset.mems_allowed = node_states[N_HIGH_MEMORY];

  mutex_unlock(&callback_mutex);
- mutex_unlock(&manage_mutex);
+ cgroup_unlock();
 }

 /*
@@ -2113,109 +1451,7 @@ void __init cpuset_init_smp(void)
 }

 /**
- * cpuset_fork - attach newly forked task to its parents cpuset.
- * @tsk: pointer to task_struct of forking parent process.
- *
- * Description: A task inherits its parent's cpuset at fork().
- *
- * A pointer to the shared cpuset was automatically copied in fork.c
- * by dup_task_struct().  However, we ignore that copy, since it was
- * not made under the protection of task_lock(), so might no longer be
- * a valid cpuset pointer.  attach_task() might have already changed
- * current->cpuset, allowing the previously referenced cpuset to
- * be removed and freed.  Instead, we task_lock(current) and copy
- * its present value of current->cpuset for our freshly forked child.
- *
- * At the point that cpuset_fork() is called, 'current' is the parent
- * task, and the passed argument 'child' points to the child task.
- **/

-void cpuset_fork(struct task_struct *child)
-{
- task_lock(current);
- child->cpuset = current->cpuset;
- atomic_inc(&child->cpuset->count);
- task_unlock(current);
-}
-
-/**
- * cpuset_exit - detach cpuset from exiting task
```

- * @tsk: pointer to task_struct of exiting process
- *
- * Description: Detach cpuset from @tsk and release it.
- *
- * Note that cpusets marked notify_on_release force every task in
- * them to take the global manage_mutex mutex when exiting.
- * This could impact scaling on very large systems. Be reluctant to
- * use notify_on_release cpusets where very high task exit scaling
- * is required on large systems.
- *
- * Don't even think about derefencing 'cs' after the cpuset use count
- * goes to zero, except inside a critical section guarded by manage_mutex
- * or callback_mutex.   Otherwise a zero cpuset use count is a license to
- * any other task to nuke the cpuset immediately, via cpuset_rmdir().
- *
- * This routine has to take manage_mutex, not callback_mutex, because
- * it is holding that mutex while calling check_for_release(),
- * which calls kmalloc(), so can't be called holding callback_mutex().
- *
- * the_top_cpuset_hack:
- *
- *   Set the exiting tasks cpuset to the root cpuset (top_cpuset).
- *
- *   Don't leave a task unable to allocate memory, as that is an
- *   accident waiting to happen should someone add a callout in
- *   do_exit() after the cpuset_exit() call that might allocate.
- *   If a task tries to allocate memory with an invalid cpuset,
- *   it will oops in cpuset_update_task_memory_state().
- *
- *   We call cpuset_exit() while the task is still competent to
- *   handle notify_on_release(), then leave the task attached to
- *   the root cpuset (top_cpuset) for the remainder of its exit.
- *
- *   To do this properly, we would increment the reference count on
- *   top_cpuset, and near the very end of the kernel/exit.c do_exit()
- *   code we would add a second cpuset function call, to drop that
- *   reference. This would just create an unnecessary hot spot on
- *   the top_cpuset reference count, to no avail.
- *
- *   Normally, holding a reference to a cpuset without bumping its
- *   count is unsafe.   The cpuset could go away, or someone could
- *   attach us to a different cpuset, decrementing the count on
- *   the first cpuset that we never incremented. But in this case,
- *   top_cpuset isn't going away, and either task has PF_EXITING set,
- *   which wards off any attach_task() attempts, or task is a failed
- *   fork, never visible to attach_task.
- *
- *   Another way to do this would be to set the cpuset pointer

```
- *    to NULL here, and check in cpuset_update_task_memory_state()
- *    for a NULL pointer.  This hack avoids that NULL check, for no
- *    cost (other than this way too long comment ;).
- **/
-
-void cpuset_exit(struct task_struct *tsk)
-{
- struct cpuset *cs;
-
- task_lock(current);
- cs = tsk->cpuset;
- tsk->cpuset = &top_cpuset; /* the_top_cpuset_hack - see above */
- task_unlock(current);
-
- if (notify_on_release(cs)) {
-  char *pathbuf = NULL;
-
-  mutex_lock(&manage_mutex);
-  if (atomic_dec_and_test(&cs->count))
-   check_for_release(cs, &pathbuf);
-  mutex_unlock(&manage_mutex);
-  cpuset_release_agent(pathbuf);
- } else {
-  atomic_dec(&cs->count);
- }
-}
-
-/**
  * cpuset_cpus_allowed - return cpus_allowed mask from a tasks cpuset.
  * @tsk: pointer to task_struct from which to obtain cpuset->cpus_allowed.
  *
@@ -2231,7 +1467,7 @@ cpumask_t cpuset_cpus_allowed(struct tas

  mutex_lock(&callback_mutex);
  task_lock(tsk);
- guarantee_online_cpus(tsk->cpuset, &mask);
+ guarantee_online_cpus(task_cs(tsk), &mask);
  task_unlock(tsk);
  mutex_unlock(&callback_mutex);

@@ -2259,7 +1495,7 @@ nodemask_t cpuset_mems_allowed(struct ta

  mutex_lock(&callback_mutex);
  task_lock(tsk);
- guarantee_online_mems(tsk->cpuset, &mask);
+ guarantee_online_mems(task_cs(tsk), &mask);
  task_unlock(tsk);
  mutex_unlock(&callback_mutex);
```

```
@@ -2390,7 +1626,7 @@ int __cpuset_zone_allowed_softwall(struc
  mutex_lock(&callback_mutex);

  task_lock(current);
- cs = nearest_exclusive_ancestor(current->cpuset);
+ cs = nearest_exclusive_ancestor(task_cs(current));
  task_unlock(current);

  allowed = node_isset(node, cs->mems_allowed);
@@ -2527,7 +1763,7 @@ int cpuset_excl_nodes_overlap(const stru
   task_unlock(current);
   goto done;
  }
- cs1 = nearest_exclusive_ancestor(current->cpuset);
+ cs1 = nearest_exclusive_ancestor(task_cs(current));
  task_unlock(current);

  task_lock((struct task_struct *)p);
@@ -2535,7 +1771,7 @@ int cpuset_excl_nodes_overlap(const stru
   task_unlock((struct task_struct *)p);
   goto done;
  }
- cs2 = nearest_exclusive_ancestor(p->cpuset);
+ cs2 = nearest_exclusive_ancestor(task_cs((struct task_struct *)p));
  task_unlock((struct task_struct *)p);

  overlap = nodes_intersects(cs1->mems_allowed, cs2->mems_allowed);
@@ -2571,14 +1807,12 @@ int cpuset_memory_pressure_enabled __rea

 void __cpuset_memory_pressure_bump(void)
 {
- struct cpuset *cs;
-
  task_lock(current);
- cs = current->cpuset;
- fmeter_markevent(&cs->fmeter);
+ fmeter_markevent(&task_cs(current)->fmeter);
  task_unlock(current);
 }

+#ifdef CONFIG_PROC_PID_CPUSET
 /*
  * proc_cpuset_show()
  *  - Print tasks cpuset path into seq_file.
@@ -2595,6 +1829,7 @@ static int proc_cpuset_show(struct seq_f
  struct pid *pid;
  struct task_struct *tsk;
```

```
  char *buf;
+ struct cgroup_subsys_state *css;
  int retval;

  retval = -ENOMEM;
@@ -2609,15 +1844,15 @@ static int proc_cpuset_show(struct seq_f
   goto out_free;

  retval = -EINVAL;
- mutex_lock(&manage_mutex);
-
- retval = cpuset_path(tsk->cpuset, buf, PAGE_SIZE);
+ cgroup_lock();
+ css = task_subsys_state(tsk, cpuset_subsys_id);
+ retval = cgroup_path(css->cgroup, buf, PAGE_SIZE);
  if (retval < 0)
   goto out_unlock;
  seq_puts(m, buf);
  seq_putc(m, '\n');
 out_unlock:
- mutex_unlock(&manage_mutex);
+ cgroup_unlock();
  put_task_struct(tsk);
 out_free:
  kfree(buf);
@@ -2637,6 +1872,7 @@ const struct file_operations proc_cpuset
  .llseek  = seq_lseek,
  .release = single_release,
 };
+#endif /* CONFIG_PROC_PID_CPUSET */

 /* Display task cpus_allowed, mems_allowed in /proc/<pid>/status file. */
 char *cpuset_task_status_allowed(struct task_struct *task, char *buffer)
diff -puN kernel/exit.c~task-cgroupsv11-make-cpusets-a-client-of-cgroups kernel/exit.c
--- a/kernel/exit.c~task-cgroupsv11-make-cpusets-a-client-of-cgroups
+++ a/kernel/exit.c
@@ -32,7 +32,6 @@
 #include <linux/taskstats_kern.h>
 #include <linux/delayacct.h>
 #include <linux/freezer.h>
-#include <linux/cpuset.h>
 #include <linux/cgroup.h>
 #include <linux/syscalls.h>
 #include <linux/signal.h>
@@ -981,7 +980,6 @@ fastcall NORET_TYPE void do_exit(long co
  __exit_fs(tsk);
  check_stack_usage();
  exit_thread();
```

```
- cpuset_exit(tsk);
  cgroup_exit(tsk, 1);
  exit_keys(tsk);

diff -puN kernel/fork.c~task-cgroupsv11-make-cpusets-a-client-of-cgroups kernel/fork.c
--- a/kernel/fork.c~task-cgroupsv11-make-cpusets-a-client-of-cgroups
+++ a/kernel/fork.c
@@ -29,7 +29,6 @@
 #include <linux/nsproxy.h>
 #include <linux/capability.h>
 #include <linux/cpu.h>
-#include <linux/cpuset.h>
 #include <linux/cgroup.h>
 #include <linux/security.h>
 #include <linux/swap.h>
@@ -1069,7 +1068,6 @@ static struct task_struct *copy_process(
 #endif
   p->io_context = NULL;
   p->audit_context = NULL;
- cpuset_fork(p);
   cgroup_fork(p);
 #ifdef CONFIG_NUMA
    p->mempolicy = mpol_copy(p->mempolicy);
@@ -1318,7 +1316,6 @@ bad_fork_cleanup_policy:
   mpol_free(p->mempolicy);
 bad_fork_cleanup_cgroup:
 #endif
- cpuset_exit(p);
   cgroup_exit(p, cgroup_callbacks_done);
   delayacct_tsk_free(p);
   if (p->binfmt)
diff -puN mm/mempolicy.c~task-cgroupsv11-make-cpusets-a-client-of-cgroups mm/mempolicy.c
--- a/mm/mempolicy.c~task-cgroupsv11-make-cpusets-a-client-of-cgroups
+++ a/mm/mempolicy.c
@@ -1323,7 +1323,6 @@ EXPORT_SYMBOL(alloc_pages_current);
   * keeps mempolicies cpuset relative after its cpuset moves.  See
   * further kernel/cpuset.c update_nodemask().
   */
-void *cpuset_being_rebound;

 /* Slow path of a mempolicy copy */
 struct mempolicy *__mpol_copy(struct mempolicy *old)
@@ -1948,4 +1947,3 @@ out:
    m->version = (vma != priv->tail_vma) ? vma->vm_start : 0;
   return 0;
 }
-

_
```

--

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 12/29] task containersv11 example cpu accounting subsystem
Posted by Paul Menage on Tue, 11 Sep 2007 19:52:51 GMT
View Forum Message <> Reply to Message

From: Paul Menage <menage@google.com>

This example demonstrates how to use the generic cgroup subsystem for a
simple resource tracker that counts, for the processes in a cgroup, the
total CPU time used and the %CPU used in the last complete 10 second interval.

Portions contributed by Balbir Singh <balbir@in.ibm.com>

Signed-off-by: Paul Menage <menage@google.com>
Cc: Serge E. Hallyn <serue@us.ibm.com>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Dave Hansen <haveblue@us.ibm.com>
Cc: Balbir Singh <balbir@in.ibm.com>
Cc: Paul Jackson <pj@sgi.com>
Cc: Kirill Korotaev <dev@openvz.org>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: Srivatsa Vaddagiri <vatsa@in.ibm.com>
Cc: Cedric Le Goater <clg@fr.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

 include/linux/cgroup_subsys.h | 6
 include/linux/cpu_acct.h | 14 ++
 init/Kconfig | 7 +
 kernel/Makefile | 1
 kernel/cpu_acct.c | 186 ++
 kernel/sched.c | 14 +-
 6 files changed, 225 insertions(+), 3 deletions(-)

diff -puN include/linux/cgroup_subsys.h~task-cgroupsv11-example-cpu-accounting-subsystem include/linux/cgroup_subsys.h
--- a/include/linux/cgroup_subsys.h~task-cgroupsv11-example-cpu-accounting-subsystem
+++ a/include/linux/cgroup_subsys.h
@@ -13,4 +13,10 @@ SUBSYS(cpuset)

 /* */

```
+#ifdef CONFIG_CGROUP_CPUACCT
+SUBSYS(cpuacct)
+#endif
+
+/* */
+
 /* */
diff -puN /dev/null include/linux/cpu_acct.h
--- /dev/null
+++ a/include/linux/cpu_acct.h
@@ -0,0 +1,14 @@
+
+#ifndef _LINUX_CPU_ACCT_H
+#define _LINUX_CPU_ACCT_H
+
+#include <linux/cgroup.h>
+#include <asm/cputime.h>
+
+#ifdef CONFIG_CGROUP_CPUACCT
+extern void cpuacct_charge(struct task_struct *, cputime_t cputime);
+#else
+static void inline cpuacct_charge(struct task_struct *p, cputime_t cputime) {}
+#endif
+
+#endif
diff -puN init/Kconfig~task-cgroupsv11-example-cpu-accounting-subsystem init/Kconfig
--- a/init/Kconfig~task-cgroupsv11-example-cpu-accounting-subsystem
+++ a/init/Kconfig
@@ -318,6 +318,13 @@ config PROC_PID_CPUSET
  depends on CPUSETS
  default y

+config CGROUP_CPUACCT
+ bool "Simple CPU accounting cgroup subsystem"
+ depends on CGROUPS
+ help
+   Provides a simple Resource Controller for monitoring the
+   total CPU consumed by the tasks in a cgroup
+
 config RELAY
  bool "Kernel->user space relay support (formerly relayfs)"
  help
diff -puN kernel/Makefile~task-cgroupsv11-example-cpu-accounting-subsystem kernel/Makefile
--- a/kernel/Makefile~task-cgroupsv11-example-cpu-accounting-subsystem
+++ a/kernel/Makefile
@@ -40,6 +40,7 @@ obj-$(CONFIG_KEXEC) += kexec.o
 obj-$(CONFIG_COMPAT) += compat.o
```

```
 obj-$(CONFIG_CGROUPS) += cgroup.o
 obj-$(CONFIG_CPUSETS) += cpuset.o
+obj-$(CONFIG_CGROUP_CPUACCT) += cpu_acct.o
 obj-$(CONFIG_IKCONFIG) += configs.o
 obj-$(CONFIG_STOP_MACHINE) += stop_machine.o
 obj-$(CONFIG_AUDIT) += audit.o auditfilter.o
diff -puN /dev/null kernel/cpu_acct.c
--- /dev/null
+++ a/kernel/cpu_acct.c
@@ -0,0 +1,186 @@
+/*
+ * kernel/cpu_acct.c - CPU accounting cgroup subsystem
+ *
+ * Copyright (C) Google Inc, 2006
+ *
+ * Developed by Paul Menage (menage@google.com) and Balbir Singh
+ * (balbir@in.ibm.com)
+ *
+ */
+
+/*
+ * Example cgroup subsystem for reporting total CPU usage of tasks in a
+ * cgroup, along with percentage load over a time interval
+ */
+
+#include <linux/module.h>
+#include <linux/cgroup.h>
+#include <linux/fs.h>
+#include <linux/rcupdate.h>
+
+#include <asm/div64.h>
+
+struct cpuacct {
+ struct cgroup_subsys_state css;
+ spinlock_t lock;
+ /* total time used by this class */
+ cputime64_t time;
+
+ /* time when next load calculation occurs */
+ u64 next_interval_check;
+
+ /* time used in current period */
+ cputime64_t current_interval_time;
+
+ /* time used in last period */
+ cputime64_t last_interval_time;
+};
+
```

```
+struct cgroup_subsys cpuacct_subsys;
+
+static inline struct cpuacct *cgroup_ca(struct cgroup *cont)
+{
+ return container_of(cgroup_subsys_state(cont, cpuacct_subsys_id),
+       struct cpuacct, css);
+}
+
+static inline struct cpuacct *task_ca(struct task_struct *task)
+{
+ return container_of(task_subsys_state(task, cpuacct_subsys_id),
+       struct cpuacct, css);
+}
+
+#define INTERVAL (HZ * 10)
+
+static inline u64 next_interval_boundary(u64 now)
+{
+ /* calculate the next interval boundary beyond the
+  * current time */
+ do_div(now, INTERVAL);
+ return (now + 1) * INTERVAL;
+}
+
+static struct cgroup_subsys_state *cpuacct_create(
+ struct cgroup_subsys *ss, struct cgroup *cont)
+{
+ struct cpuacct *ca = kzalloc(sizeof(*ca), GFP_KERNEL);
+
+ if (!ca)
+  return ERR_PTR(-ENOMEM);
+ spin_lock_init(&ca->lock);
+ ca->next_interval_check = next_interval_boundary(get_jiffies_64());
+ return &ca->css;
+}
+
+static void cpuacct_destroy(struct cgroup_subsys *ss,
+       struct cgroup *cont)
+{
+ kfree(cgroup_ca(cont));
+}
+
+/* Lazily update the load calculation if necessary. Called with ca locked */
+static void cpuusage_update(struct cpuacct *ca)
+{
+ u64 now = get_jiffies_64();
+
+ /* If we're not due for an update, return */
```

```
+ if (ca->next_interval_check > now)
+  return;
+
+ if (ca->next_interval_check <= (now - INTERVAL)) {
+  /* If it's been more than an interval since the last
+   * check, then catch up - the last interval must have
+   * been zero load */
+  ca->last_interval_time = 0;
+  ca->next_interval_check = next_interval_boundary(now);
+ } else {
+  /* If a steal takes the last interval time negative,
+   * then we just ignore it */
+  if ((s64)ca->current_interval_time > 0)
+   ca->last_interval_time = ca->current_interval_time;
+  else
+   ca->last_interval_time = 0;
+  ca->next_interval_check += INTERVAL;
+ }
+ ca->current_interval_time = 0;
+}
+
+static u64 cpuusage_read(struct cgroup *cont, struct cftype *cft)
+{
+ struct cpuacct *ca = cgroup_ca(cont);
+ u64 time;
+
+ spin_lock_irq(&ca->lock);
+ cpuusage_update(ca);
+ time = cputime64_to_jiffies64(ca->time);
+ spin_unlock_irq(&ca->lock);
+
+ /* Convert 64-bit jiffies to seconds */
+ time *= 1000;
+ do_div(time, HZ);
+ return time;
+}
+
+static u64 load_read(struct cgroup *cont, struct cftype *cft)
+{
+ struct cpuacct *ca = cgroup_ca(cont);
+ u64 time;
+
+ /* Find the time used in the previous interval */
+ spin_lock_irq(&ca->lock);
+ cpuusage_update(ca);
+ time = cputime64_to_jiffies64(ca->last_interval_time);
+ spin_unlock_irq(&ca->lock);
+
```

```
+ /* Convert time to a percentage, to give the load in the
+  * previous period */
+ time *= 100;
+ do_div(time, INTERVAL);
+
+ return time;
+}
+
+static struct cftype files[] = {
+ {
+  .name = "usage",
+  .read_uint = cpuusage_read,
+ },
+ {
+  .name = "load",
+  .read_uint = load_read,
+ }
+};
+
+static int cpuacct_populate(struct cgroup_subsys *ss, struct cgroup *cont)
+{
+ return cgroup_add_files(cont, ss, files, ARRAY_SIZE(files));
+}
+
+void cpuacct_charge(struct task_struct *task, cputime_t cputime)
+{
+
+ struct cpuacct *ca;
+ unsigned long flags;
+
+ if (!cpuacct_subsys.active)
+  return;
+ rcu_read_lock();
+ ca = task_ca(task);
+ if (ca) {
+  spin_lock_irqsave(&ca->lock, flags);
+  cpuusage_update(ca);
+  ca->time = cputime64_add(ca->time, cputime);
+  ca->current_interval_time =
+   cputime64_add(ca->current_interval_time, cputime);
+  spin_unlock_irqrestore(&ca->lock, flags);
+ }
+ rcu_read_unlock();
+}
+
+struct cgroup_subsys cpuacct_subsys = {
+ .name = "cpuacct",
+ .create = cpuacct_create,
```

```
+ .destroy = cpuacct_destroy,
+ .populate = cpuacct_populate,
+ .subsys_id = cpuacct_subsys_id,
+};
diff -puN kernel/sched.c~task-cgroupsv11-example-cpu-accounting-subsystem kernel/sched.c
--- a/kernel/sched.c~task-cgroupsv11-example-cpu-accounting-subsystem
+++ a/kernel/sched.c
@@ -51,6 +51,7 @@
 #include <linux/cpu.h>
 #include <linux/cpuset.h>
 #include <linux/percpu.h>
+#include <linux/cpu_acct.h>
 #include <linux/kthread.h>
 #include <linux/seq_file.h>
 #include <linux/sysctl.h>
@@ -3268,9 +3269,13 @@ void account_user_time(struct task_struc
 {
  struct cpu_usage_stat *cpustat = &kstat_this_cpu.cpustat;
  cputime64_t tmp;
+ struct rq *rq = this_rq();

  p->utime = cputime_add(p->utime, cputime);

+ if (p != rq->idle)
+  cpuacct_charge(p, cputime);
+
  /* Add user time to cpustat. */
  tmp = cputime_to_cputime64(cputime);
  if (TASK_NICE(p) > 0)
@@ -3300,9 +3305,10 @@ void account_system_time(struct task_str
   cpustat->irq = cputime64_add(cpustat->irq, tmp);
  else if (softirq_count())
   cpustat->softirq = cputime64_add(cpustat->softirq, tmp);
- else if (p != rq->idle)
+ else if (p != rq->idle) {
   cpustat->system = cputime64_add(cpustat->system, tmp);
- else if (atomic_read(&rq->nr_iowait) > 0)
+  cpuacct_charge(p, cputime);
+ } else if (atomic_read(&rq->nr_iowait) > 0)
   cpustat->iowait = cputime64_add(cpustat->iowait, tmp);
  else
   cpustat->idle = cputime64_add(cpustat->idle, tmp);
@@ -3327,8 +3333,10 @@ void account_steal_time(struct task_stru
    cpustat->iowait = cputime64_add(cpustat->iowait, tmp);
   else
    cpustat->idle = cputime64_add(cpustat->idle, tmp);
- } else
+ } else {
```

```
   cpustat->steal = cputime64_add(cpustat->steal, tmp);
+  cpuacct_charge(p, -tmp);
+ }
 }

 /*

_

--
```

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 13/29] task containersv11 simple task container debug info subsystem

Posted by Paul Menage on Tue, 11 Sep 2007 19:52:52 GMT
View Forum Message <> Reply to Message

From: Paul Menage <menage@google.com>

This example subsystem exports debugging information as an aid to diagnosing
refcount leaks, etc, in the cgroup framework.

Signed-off-by: Paul Menage <menage@google.com>
Cc: Serge E. Hallyn <serue@us.ibm.com>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Dave Hansen <haveblue@us.ibm.com>
Cc: Balbir Singh <balbir@in.ibm.com>
Cc: Paul Jackson <pj@sgi.com>
Cc: Kirill Korotaev <dev@openvz.org>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: Srivatsa Vaddagiri <vatsa@in.ibm.com>
Cc: Cedric Le Goater <clg@fr.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>

```
 include/linux/cgroup_subsys.h |    4 +
 init/Kconfig                  |   10 ++
 kernel/Makefile               |    1
 kernel/cgroup_debug.c         |   97 ++++++++++++++++++++++++++++++++++++++++++
 4 files changed, 112 insertions(+)

diff -puN
include/linux/cgroup_subsys.h~task-cgroupsv11-simple-task-cgroup-debug-info-subsystem
include/linux/cgroup_subsys.h
--- a/include/linux/cgroup_subsys.h~task-cgroupsv11-simple-task-cgroup-debug-info-subsystem
```

```
+++ a/include/linux/cgroup_subsys.h
@@ -19,4 +19,8 @@ SUBSYS(cpuacct)

 /* */

+#ifdef CONFIG_CGROUP_DEBUG
+SUBSYS(debug)
+#endif
+
 /* */
diff -puN init/Kconfig~task-cgroupsv11-simple-task-cgroup-debug-info-subsystem init/Kconfig
--- a/init/Kconfig~task-cgroupsv11-simple-task-cgroup-debug-info-subsystem
+++ a/init/Kconfig
@@ -282,6 +282,16 @@ config CGROUPS

    Say N if unsure.

+config CGROUP_DEBUG
+ bool "Example debug cgroup subsystem"
+ depends on CGROUPS
+ help
+   This option enables a simple cgroup subsystem that
+   exports useful debugging information about the cgroups
+   framework
+
+   Say N if unsure
+
 config CPUSETS
  bool "Cpuset support"
  depends on SMP && CGROUPS
diff -puN kernel/Makefile~task-cgroupsv11-simple-task-cgroup-debug-info-subsystem
kernel/Makefile
--- a/kernel/Makefile~task-cgroupsv11-simple-task-cgroup-debug-info-subsystem
+++ a/kernel/Makefile
@@ -39,6 +39,7 @@ obj-$(CONFIG_BSD_PROCESS_ACCT) += acct.o
 obj-$(CONFIG_KEXEC) += kexec.o
 obj-$(CONFIG_COMPAT) += compat.o
 obj-$(CONFIG_CGROUPS) += cgroup.o
+obj-$(CONFIG_CGROUP_DEBUG) += cgroup_debug.o
 obj-$(CONFIG_CPUSETS) += cpuset.o
 obj-$(CONFIG_CGROUP_CPUACCT) += cpu_acct.o
 obj-$(CONFIG_IKCONFIG) += configs.o
diff -puN /dev/null kernel/cgroup_debug.c
--- /dev/null
+++ a/kernel/cgroup_debug.c
@@ -0,0 +1,97 @@
+/*
+ * kernel/ccontainer_debug.c - Example cgroup subsystem that
```

```
+ * exposes debug info
+ *
+ * Copyright (C) Google Inc, 2007
+ *
+ * Developed by Paul Menage (menage@google.com)
+ *
+ */
+
+#include <linux/cgroup.h>
+#include <linux/fs.h>
+#include <linux/slab.h>
+#include <linux/rcupdate.h>
+
+#include <asm/atomic.h>
+
+static struct cgroup_subsys_state *debug_create(struct cgroup_subsys *ss,
+         struct cgroup *cont)
+{
+ struct cgroup_subsys_state *css = kzalloc(sizeof(*css), GFP_KERNEL);
+
+ if (!css)
+  return ERR_PTR(-ENOMEM);
+
+ return css;
+}
+
+static void debug_destroy(struct cgroup_subsys *ss, struct cgroup *cont)
+{
+ kfree(cont->subsys[debug_subsys_id]);
+}
+
+static u64 cgroup_refcount_read(struct cgroup *cont, struct cftype *cft)
+{
+ return atomic_read(&cont->count);
+}
+
+static u64 taskcount_read(struct cgroup *cont, struct cftype *cft)
+{
+ u64 count;
+
+ cgroup_lock();
+ count = cgroup_task_count(cont);
+ cgroup_unlock();
+ return count;
+}
+
+static u64 current_css_set_read(struct cgroup *cont, struct cftype *cft)
+{
```

```
+ return (u64)(long)current->cgroups;
+}
+
+static u64 current_css_set_refcount_read(struct cgroup *cont,
+        struct cftype *cft)
+{
+ u64 count;
+
+ rcu_read_lock();
+ count = atomic_read(&current->cgroups->ref.refcount);
+ rcu_read_unlock();
+ return count;
+}
+
+static struct cftype files[] =  {
+ {
+  .name = "cgroup_refcount",
+  .read_uint = cgroup_refcount_read,
+ },
+ {
+  .name = "taskcount",
+  .read_uint = taskcount_read,
+ },
+
+ {
+  .name = "current_css_set",
+  .read_uint = current_css_set_read,
+ },
+
+ {
+  .name = "current_css_set_refcount",
+  .read_uint = current_css_set_refcount_read,
+ },
+};
+
+static int debug_populate(struct cgroup_subsys *ss, struct cgroup *cont)
+{
+ return cgroup_add_files(cont, ss, files, ARRAY_SIZE(files));
+}
+
+struct cgroup_subsys debug_subsys = {
+ .name = "debug",
+ .create = debug_create,
+ .destroy = debug_destroy,
+ .populate = debug_populate,
+ .subsys_id = debug_subsys_id,
+};
_
```

--

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 14/29] add containerstats v3
Posted by Paul Menage on Tue, 11 Sep 2007 19:52:53 GMT
View Forum Message <> Reply to Message

From: Balbir Singh <balbir@linux.vnet.ibm.com>

This patch is inspired by the discussion at
http://lkml.org/lkml/2007/4/11/187 and implements per cgroup statistics
as suggested by Andrew Morton in http://lkml.org/lkml/2007/4/11/263. The
patch is on top of 2.6.21-mm1 with Paul's cgroups v9 patches (forward
ported)

This patch implements per cgroup statistics infrastructure and re-uses
code from the taskstats interface. A new set of cgroup operations are
registered with commands and attributes. It should be very easy to
extend per cgroup statistics, by adding members to the cgroupstats
structure.

The current model for cgroupstats is a pull, a push model (to post
statistics on interesting events), should be very easy to add. Currently
user space requests for statistics by passing the cgroup file
descriptor. Statistics about the state of all the tasks in the cgroup
is returned to user space.

TODO's/NOTE:

This patch provides an infrastructure for implementing cgroup statistics.
Based on the needs of each controller, we can incrementally add more statistics,
event based support for notification of statistics, accumulation of taskstats
into cgroup statistics in the future.

Sample output

./cgroupstats -C /cgroup/a
sleeping 2, blocked 0, running 1, stopped 0, uninterruptible 0

./cgroupstats -C /cgroup/
sleeping 154, blocked 0, running 0, stopped 0, uninterruptible 0

If the approach looks good, I'll enhance and post the user space utility for

the same

Feedback, comments, test results are always welcome!

Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>
Cc: Paul Menage <menage@google.com>
Cc: Jay Lan <jlan@engr.sgi.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>

```
 Documentation/accounting/cgroupstats.txt |   27 ++++++
 include/linux/Kbuild                     |    1
 include/linux/cgroup.h                   |    8 ++
 include/linux/cgroupstats.h              |   70 ++++++++++++++++++++
 include/linux/delayacct.h                |   13 +++
 kernel/cgroup.c                          |   55 ++++++++++++++++
 kernel/taskstats.c                       |   66 ++++++++++++++++++
 7 files changed, 240 insertions(+)
```

```
diff -puN /dev/null Documentation/accounting/cgroupstats.txt
--- /dev/null
+++ a/Documentation/accounting/cgroupstats.txt
@@ -0,0 +1,27 @@
+Control Groupstats is inspired by the discussion at
+http://lkml.org/lkml/2007/4/11/187 and implements per cgroup statistics as
+suggested by Andrew Morton in http://lkml.org/lkml/2007/4/11/263.
+
+Per cgroup statistics infrastructure re-uses code from the taskstats
+interface. A new set of cgroup operations are registered with commands
+and attributes specific to cgroups. It should be very easy to
+extend per cgroup statistics, by adding members to the cgroupstats
+structure.
+
+The current model for cgroupstats is a pull, a push model (to post
+statistics on interesting events), should be very easy to add. Currently
+user space requests for statistics by passing the cgroup path.
+Statistics about the state of all the tasks in the cgroup is returned to
+user space.
+
+NOTE: We currently rely on delay accounting for extracting information
+about tasks blocked on I/O. If CONFIG_TASK_DELAY_ACCT is disabled, this
+information will not be available.
+
+To extract cgroup statistics a utility very similar to getdelays.c
+has been developed, the sample output of the utility is shown below
+
+~/balbir/cgroupstats # ./getdelays  -C "/cgroup/a"
+sleeping 1, blocked 0, running 1, stopped 0, uninterruptible 0
```

```
+~/balbir/cgroupstats # ./getdelays  -C "/cgroup"
+sleeping 155, blocked 0, running 1, stopped 0, uninterruptible 2
diff -puN include/linux/Kbuild~add-cgroupstats-v3 include/linux/Kbuild
--- a/include/linux/Kbuild~add-cgroupstats-v3
+++ a/include/linux/Kbuild
@@ -47,6 +47,7 @@ header-y += coda_psdev.h
 header-y += coff.h
 header-y += comstats.h
 header-y += const.h
+header-y += cgroupstats.h
 header-y += cycx_cfm.h
 header-y += dlm_device.h
 header-y += dlm_netlink.h
diff -puN include/linux/cgroup.h~add-cgroupstats-v3 include/linux/cgroup.h
--- a/include/linux/cgroup.h~add-cgroupstats-v3
+++ a/include/linux/cgroup.h
@@ -13,6 +13,7 @@
 #include <linux/cpumask.h>
 #include <linux/nodemask.h>
 #include <linux/rcupdate.h>
+#include <linux/cgroupstats.h>

 #ifdef CONFIG_CGROUPS

@@ -29,6 +30,8 @@ extern void cgroup_fork(struct task_s
 extern void cgroup_fork_callbacks(struct task_struct *p);
 extern void cgroup_post_fork(struct task_struct *p);
 extern void cgroup_exit(struct task_struct *p, int run_callbacks);
+extern int cgroupstats_build(struct cgroupstats *stats,
+    struct dentry *dentry);

 extern struct file_operations proc_cgroup_operations;

@@ -305,6 +308,11 @@ static inline void cgroup_exit(struct

 static inline void cgroup_lock(void) {}
 static inline void cgroup_unlock(void) {}
+static inline int cgroupstats_build(struct cgroupstats *stats,
+     struct dentry *dentry)
+{
+ return -EINVAL;
+}

 #endif /* !CONFIG_CGROUPS */

diff -puN /dev/null include/linux/cgroupstats.h
--- /dev/null
+++ a/include/linux/cgroupstats.h
```

```
@@ -0,0 +1,70 @@
+/* cgroupstats.h - exporting per-cgroup statistics
+ *
+ * Copyright IBM Corporation, 2007
+ * Author Balbir Singh <balbir@linux.vnet.ibm.com>
+ *
+ * This program is free software; you can redistribute it and/or modify it
+ * under the terms of version 2.1 of the GNU Lesser General Public License
+ * as published by the Free Software Foundation.
+ *
+ * This program is distributed in the hope that it would be useful, but
+ * WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
+ */
+
+#ifndef _LINUX_CGROUPSTATS_H
+#define _LINUX_CGROUPSTATS_H
+
+#include <linux/taskstats.h>
+
+/*
+ * Data shared between user space and kernel space on a per cgroup
+ * basis. This data is shared using taskstats.
+ *
+ * Most of these states are derived by looking at the task->state value
+ * For the nr_io_wait state, a flag in the delay accounting structure
+ * indicates that the task is waiting on IO
+ *
+ * Each member is aligned to a 8 byte boundary.
+ */
+struct cgroupstats {
+ __u64 nr_sleeping;  /* Number of tasks sleeping */
+ __u64 nr_running;  /* Number of tasks running */
+ __u64 nr_stopped;  /* Number of tasks in stopped state */
+ __u64 nr_uninterruptible; /* Number of tasks in uninterruptible */
+     /* state */
+ __u64 nr_io_wait;  /* Number of tasks waiting on IO */
+};
+
+/*
+ * Commands sent from userspace
+ * Not versioned. New commands should only be inserted at the enum's end
+ * prior to __CGROUPSTATS_CMD_MAX
+ */
+
+enum {
+ CGROUPSTATS_CMD_UNSPEC = __TASKSTATS_CMD_MAX, /* Reserved */
+ CGROUPSTATS_CMD_GET,  /* user->kernel request/get-response */
```

```
+ CGROUPSTATS_CMD_NEW,  /* kernel->user event */
+ __CGROUPSTATS_CMD_MAX,
+};
+
+#define CGROUPSTATS_CMD_MAX (__CGROUPSTATS_CMD_MAX - 1)
+
+enum {
+ CGROUPSTATS_TYPE_UNSPEC = 0, /* Reserved */
+ CGROUPSTATS_TYPE_CGROUP_STATS, /* contains name + stats */
+ __CGROUPSTATS_TYPE_MAX,
+};
+
+#define CGROUPSTATS_TYPE_MAX (__CGROUPSTATS_TYPE_MAX - 1)
+
+enum {
+ CGROUPSTATS_CMD_ATTR_UNSPEC = 0,
+ CGROUPSTATS_CMD_ATTR_FD,
+ __CGROUPSTATS_CMD_ATTR_MAX,
+};
+
+#define CGROUPSTATS_CMD_ATTR_MAX (__CGROUPSTATS_CMD_ATTR_MAX - 1)
+
+#endif /* _LINUX_CGROUPSTATS_H */
diff -puN include/linux/delayacct.h~add-cgroupstats-v3 include/linux/delayacct.h
--- a/include/linux/delayacct.h~add-cgroupstats-v3
+++ a/include/linux/delayacct.h
@@ -26,6 +26,7 @@
  * Used to set current->delays->flags
  */
 #define DELAYACCT_PF_SWAPIN 0x00000001 /* I am doing a swapin */
+#define DELAYACCT_PF_BLKIO 0x00000002 /* I am waiting on IO */

 #ifdef CONFIG_TASK_DELAY_ACCT

@@ -39,6 +40,14 @@ extern void __delayacct_blkio_end(void);
 extern int __delayacct_add_tsk(struct taskstats *, struct task_struct *);
 extern __u64 __delayacct_blkio_ticks(struct task_struct *);

+static inline int delayacct_is_task_waiting_on_io(struct task_struct *p)
+{
+ if (p->delays)
+  return (p->delays->flags & DELAYACCT_PF_BLKIO);
+ else
+  return 0;
+}
+
 static inline void delayacct_set_flag(int flag)
 {
```

```
  if (current->delays)
@@ -71,6 +80,7 @@ static inline void delayacct_tsk_free(st

 static inline void delayacct_blkio_start(void)
 {
+ delayacct_set_flag(DELAYACCT_PF_BLKIO);
  if (current->delays)
   __delayacct_blkio_start();
 }
@@ -79,6 +89,7 @@ static inline void delayacct_blkio_end(v
 {
  if (current->delays)
   __delayacct_blkio_end();
+ delayacct_clear_flag(DELAYACCT_PF_BLKIO);
 }

 static inline int delayacct_add_tsk(struct taskstats *d,
@@ -116,6 +127,8 @@ static inline int delayacct_add_tsk(stru
 { return 0; }
 static inline __u64 delayacct_blkio_ticks(struct task_struct *tsk)
 { return 0; }
+static inline int delayacct_is_task_waiting_on_io(struct task_struct *p)
+{ return 0; }
 #endif /* CONFIG_TASK_DELAY_ACCT */

 #endif
diff -puN kernel/cgroup.c~add-cgroupstats-v3 kernel/cgroup.c
--- a/kernel/cgroup.c~add-cgroupstats-v3
+++ a/kernel/cgroup.c
@@ -42,6 +42,9 @@
 #include <linux/spinlock.h>
 #include <linux/string.h>
 #include <linux/sort.h>
+#include <linux/delayacct.h>
+#include <linux/cgroupstats.h>
+
 #include <asm/atomic.h>

 static DEFINE_MUTEX(cgroup_mutex);
@@ -1739,6 +1742,58 @@ static int pid_array_load(pid_t *pidarra
  return n;
 }

+/**
+ * Build and fill cgroupstats so that taskstats can export it to user
+ * space.
+ *
+ * @stats: cgroupstats to fill information into
```

```
+ * @dentry: A dentry entry belonging to the cgroup for which stats have
+ * been requested.
+ */
+int cgroupstats_build(struct cgroupstats *stats, struct dentry *dentry)
+{
+ int ret = -EINVAL;
+ struct cgroup *cont;
+ struct cgroup_iter it;
+ struct task_struct *tsk;
+ /*
+  * Validate dentry by checking the superblock operations
+  */
+ if (dentry->d_sb->s_op != &cgroup_ops)
+   goto err;
+
+ ret = 0;
+ cont = dentry->d_fsdata;
+ rcu_read_lock();
+
+ cgroup_iter_start(cont, &it);
+ while ((tsk = cgroup_iter_next(cont, &it))) {
+  switch (tsk->state) {
+  case TASK_RUNNING:
+   stats->nr_running++;
+   break;
+  case TASK_INTERRUPTIBLE:
+   stats->nr_sleeping++;
+   break;
+  case TASK_UNINTERRUPTIBLE:
+   stats->nr_uninterruptible++;
+   break;
+  case TASK_STOPPED:
+   stats->nr_stopped++;
+   break;
+  default:
+   if (delayacct_is_task_waiting_on_io(tsk))
+    stats->nr_io_wait++;
+   break;
+  }
+ }
+ cgroup_iter_end(cont, &it);
+
+ rcu_read_unlock();
+err:
+ return ret;
+}
+
 static int cmppid(const void *a, const void *b)
```

```
 {
  return *(pid_t *)a - *(pid_t *)b;
diff -puN kernel/taskstats.c~add-cgroupstats-v3 kernel/taskstats.c
--- a/kernel/taskstats.c~add-cgroupstats-v3
+++ a/kernel/taskstats.c
@@ -22,6 +22,9 @@
 #include <linux/delayacct.h>
 #include <linux/cpumask.h>
 #include <linux/percpu.h>
+#include <linux/cgroupstats.h>
+#include <linux/cgroup.h>
+#include <linux/file.h>
 #include <net/genetlink.h>
 #include <asm/atomic.h>

@@ -49,6 +52,11 @@ __read_mostly = {
  [TASKSTATS_CMD_ATTR_REGISTER_CPUMASK] = { .type = NLA_STRING },
  [TASKSTATS_CMD_ATTR_DEREGISTER_CPUMASK] = { .type = NLA_STRING },};

+static struct nla_policy
+cgroupstats_cmd_get_policy[CGROUPSTATS_CMD_ATTR_MAX+1] __read_mostly = {
+ [CGROUPSTATS_CMD_ATTR_FD] = { .type = NLA_U32 },
+};
+
 struct listener {
  struct list_head list;
  pid_t pid;
@@ -374,6 +382,51 @@ err:
  return NULL;
 }

+static int cgroupstats_user_cmd(struct sk_buff *skb, struct genl_info *info)
+{
+ int rc = 0;
+ struct sk_buff *rep_skb;
+ struct cgroupstats *stats;
+ struct nlattr *na;
+ size_t size;
+ u32 fd;
+ struct file *file;
+ int fput_needed;
+
+ na = info->attrs[CGROUPSTATS_CMD_ATTR_FD];
+ if (!na)
+  return -EINVAL;
+
+ fd = nla_get_u32(info->attrs[CGROUPSTATS_CMD_ATTR_FD]);
+ file = fget_light(fd, &fput_needed);
```

```
+ if (file) {
+  size = nla_total_size(sizeof(struct cgroupstats));
+
+  rc = prepare_reply(info, CGROUPSTATS_CMD_NEW, &rep_skb,
+     size);
+  if (rc < 0)
+   goto err;
+
+  na = nla_reserve(rep_skb, CGROUPSTATS_TYPE_CGROUP_STATS,
+     sizeof(struct cgroupstats));
+  stats = nla_data(na);
+  memset(stats, 0, sizeof(*stats));
+
+  rc = cgroupstats_build(stats, file->f_dentry);
+  if (rc < 0)
+   goto err;
+
+  fput_light(file, fput_needed);
+  return send_reply(rep_skb, info->snd_pid);
+ }
+
+err:
+ if (file)
+  fput_light(file, fput_needed);
+ nlmsg_free(rep_skb);
+ return rc;
+}
+
 static int taskstats_user_cmd(struct sk_buff *skb, struct genl_info *info)
 {
  int rc = 0;
@@ -524,6 +577,12 @@ static struct genl_ops taskstats_ops = {
  .policy  = taskstats_cmd_get_policy,
 };

+static struct genl_ops cgroupstats_ops = {
+ .cmd  = CGROUPSTATS_CMD_GET,
+ .doit  = cgroupstats_user_cmd,
+ .policy  = cgroupstats_cmd_get_policy,
+};
+
 /* Needed early in initialization */
 void __init taskstats_init_early(void)
 {
@@ -548,8 +607,15 @@ static int __init taskstats_init(void)
  if (rc < 0)
   goto err;
```

```
+ rc = genl_register_ops(&family, &cgroupstats_ops);
+ if (rc < 0)
+  goto err_cgroup_ops;
+
  family_registered = 1;
+ printk("registered taskstats version %d\n", TASKSTATS_GENL_VERSION);
  return 0;
+err_cgroup_ops:
+ genl_unregister_ops(&family, &taskstats_ops);
 err:
  genl_unregister_family(&family);
  return rc;
_

--
```

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 15/29] add containerstats v3 fix
Posted by Paul Menage on Tue, 11 Sep 2007 19:52:54 GMT
View Forum Message <> Reply to Message

From: Andrew Morton <akpm@linux-foundation.org>

kernel/taskstats.c: In function 'cgroupstats_user_cmd':
kernel/taskstats.c:414: error: dereferencing pointer to incomplete type

Cc: Paul Menage <menage@google.com>
Cc: Jay Lan <jlan@engr.sgi.com>
Cc: Balbir Singh <balbir@linux.vnet.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

 kernel/taskstats.c | 1 +
 1 file changed, 1 insertion(+)

```
diff -puN kernel/taskstats.c~add-cgroupstats-v3-fix kernel/taskstats.c
--- a/kernel/taskstats.c~add-cgroupstats-v3-fix
+++ a/kernel/taskstats.c
@@ -24,6 +24,7 @@
 #include <linux/percpu.h>
 #include <linux/cgroupstats.h>
 #include <linux/cgroup.h>
+#include <linux/fs.h>
 #include <linux/file.h>
```

#include <net/genetlink.h>
#include <asm/atomic.h>

_

--

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 16/29] containers implement namespace tracking subsystem
Posted by Paul Menage on Tue, 11 Sep 2007 19:52:55 GMT
View Forum Message <> Reply to Message

From: "Serge E. Hallyn" <serue@us.ibm.com>

When a task enters a new namespace via a clone() or unshare(), a new cgroup
is created and the task moves into it.

This version names cgroups which are automatically created using
cgroup_clone() as "node_<pid>" where pid is the pid of the unsharing or
cloned process. (Thanks Pavel for the idea) This is safe because if the
process unshares again, it will create

/cgroups/(...)/node_<pid>/node_<pid>

The only possibilities (AFAICT) for a -EEXIST on unshare are

1. pid wraparound
2. a process fails an unshare, then tries again.

Case 1 is unlikely enough that I ignore it (at least for now). In case 2, the
node_<pid> will be empty and can be rmdir'ed to make the subsequent unshare()
succeed.

Changelog:
Name cloned cgroups as "node_<pid>".

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
Cc: Paul Menage <menage@google.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

include/linux/cgroup_subsys.h | 6 +
include/linux/nsproxy.h | 7 ++
init/Kconfig | 9 ++
kernel/Makefile | 1

```
 kernel/ns_cgroup.c           |  100 +++++++++++++++++++++++++++++
 kernel/nsproxy.c             |   17 ++++
 6 files changed, 139 insertions(+), 1 deletion(-)

diff -puN include/linux/cgroup_subsys.h~cgroups-implement-namespace-tracking-subsystem include/linux/cgroup_subsys.h
--- a/include/linux/cgroup_subsys.h~cgroups-implement-namespace-tracking-subsystem
+++ a/include/linux/cgroup_subsys.h
@@ -24,3 +24,9 @@ SUBSYS(debug)
 #endif

 /* */
+
+#ifdef CONFIG_CGROUP_NS
+SUBSYS(ns)
+#endif
+
+/* */
diff -puN include/linux/nsproxy.h~cgroups-implement-namespace-tracking-subsystem include/linux/nsproxy.h
--- a/include/linux/nsproxy.h~cgroups-implement-namespace-tracking-subsystem
+++ a/include/linux/nsproxy.h
@@ -55,4 +55,11 @@ static inline void exit_task_namespaces(
   put_nsproxy(ns);
  }
 }
+
+#ifdef CONFIG_CGROUP_NS
+int ns_cgroup_clone(struct task_struct *tsk);
+#else
+static inline int ns_cgroup_clone(struct task_struct *tsk) { return 0; }
+#endif
+
 #endif
diff -puN init/Kconfig~cgroups-implement-namespace-tracking-subsystem init/Kconfig
--- a/init/Kconfig~cgroups-implement-namespace-tracking-subsystem
+++ a/init/Kconfig
@@ -323,6 +323,15 @@ config SYSFS_DEPRECATED
    If you are using a distro that was released in 2006 or later,
    it should be safe to say N here.

+config CGROUP_NS
+       bool "Namespace cgroup subsystem"
+       select CGROUPS
+       help
+         Provides a simple namespace cgroup subsystem to
+         provide hierarchical naming of sets of namespaces,
+         for instance virtual servers and checkpoint/restart
```

```
+          jobs.
+
 config PROC_PID_CPUSET
  bool "Include legacy /proc/<pid>/cpuset file"
  depends on CPUSETS
diff -puN kernel/Makefile~cgroups-implement-namespace-tracking-subsystem kernel/Makefile
--- a/kernel/Makefile~cgroups-implement-namespace-tracking-subsystem
+++ a/kernel/Makefile
@@ -42,6 +42,7 @@ obj-$(CONFIG_CGROUPS) += cgroup.o
 obj-$(CONFIG_CGROUP_DEBUG) += cgroup_debug.o
 obj-$(CONFIG_CPUSETS) += cpuset.o
 obj-$(CONFIG_CGROUP_CPUACCT) += cpu_acct.o
+obj-$(CONFIG_CGROUP_NS) += ns_cgroup.o
 obj-$(CONFIG_IKCONFIG) += configs.o
 obj-$(CONFIG_STOP_MACHINE) += stop_machine.o
 obj-$(CONFIG_AUDIT) += audit.o auditfilter.o
diff -puN /dev/null kernel/ns_cgroup.c
--- /dev/null
+++ a/kernel/ns_cgroup.c
@@ -0,0 +1,100 @@
+/*
+ * ns_cgroup.c - namespace cgroup subsystem
+ *
+ * Copyright 2006, 2007 IBM Corp
+ */
+
+#include <linux/module.h>
+#include <linux/cgroup.h>
+#include <linux/fs.h>
+
+struct ns_cgroup {
+ struct cgroup_subsys_state css;
+ spinlock_t lock;
+};
+
+struct cgroup_subsys ns_subsys;
+
+static inline struct ns_cgroup *cgroup_to_ns(
+  struct cgroup *cgroup)
+{
+ return container_of(cgroup_subsys_state(cgroup, ns_subsys_id),
+       struct ns_cgroup, css);
+}
+
+int ns_cgroup_clone(struct task_struct *task)
+{
+ return cgroup_clone(task, &ns_subsys);
+}
```

```
+
+/*
+ * Rules:
+ *   1. you can only enter a cgroup which is a child of your current
+ *      cgroup
+ *   2. you can only place another process into a cgroup if
+ *      a. you have CAP_SYS_ADMIN
+ *      b. your cgroup is an ancestor of task's destination cgroup
+ *         (hence either you are in the same cgroup as task, or in an
+ *          ancestor cgroup thereof)
+ */
+static int ns_can_attach(struct cgroup_subsys *ss,
+  struct cgroup *new_cgroup, struct task_struct *task)
+{
+ struct cgroup *orig;
+
+ if (current != task) {
+  if (!capable(CAP_SYS_ADMIN))
+   return -EPERM;
+
+  if (!cgroup_is_descendant(new_cgroup))
+   return -EPERM;
+ }
+
+ if (atomic_read(&new_cgroup->count) != 0)
+  return -EPERM;
+
+ orig = task_cgroup(task, ns_subsys_id);
+ if (orig && orig != new_cgroup->parent)
+  return -EPERM;
+
+ return 0;
+}
+
+/*
+ * Rules: you can only create a cgroup if
+ *      1. you are capable(CAP_SYS_ADMIN)
+ *      2. the target cgroup is a descendant of your own cgroup
+ */
+static struct cgroup_subsys_state *ns_create(struct cgroup_subsys *ss,
+       struct cgroup *cgroup)
+{
+ struct ns_cgroup *ns_cgroup;
+
+ if (!capable(CAP_SYS_ADMIN))
+  return ERR_PTR(-EPERM);
+ if (!cgroup_is_descendant(cgroup))
+  return ERR_PTR(-EPERM);
```

```
+
+ ns_cgroup = kzalloc(sizeof(*ns_cgroup), GFP_KERNEL);
+ if (!ns_cgroup)
+  return ERR_PTR(-ENOMEM);
+ spin_lock_init(&ns_cgroup->lock);
+ return &ns_cgroup->css;
+}
+
+static void ns_destroy(struct cgroup_subsys *ss,
+   struct cgroup *cgroup)
+{
+ struct ns_cgroup *ns_cgroup;
+
+ ns_cgroup = cgroup_to_ns(cgroup);
+ kfree(ns_cgroup);
+}
+
+struct cgroup_subsys ns_subsys = {
+ .name = "ns",
+ .can_attach = ns_can_attach,
+ .create = ns_create,
+ .destroy  = ns_destroy,
+ .subsys_id = ns_subsys_id,
+};
diff -puN kernel/nsproxy.c~cgroups-implement-namespace-tracking-subsystem kernel/nsproxy.c
--- a/kernel/nsproxy.c~cgroups-implement-namespace-tracking-subsystem
+++ a/kernel/nsproxy.c
@@ -146,7 +146,14 @@ int copy_namespaces(unsigned long flags,
   goto out;
  }

+ err = ns_cgroup_clone(tsk);
+ if (err) {
+  put_nsproxy(new_ns);
+  goto out;
+ }
+
  tsk->nsproxy = new_ns;
+
 out:
  put_nsproxy(old_ns);
  return err;
@@ -185,8 +192,16 @@ int unshare_nsproxy_namespaces(unsigned

  *new_nsp = create_new_namespaces(unshare_flags, current,
     new_fs ? new_fs : current->fs);
- if (IS_ERR(*new_nsp))
+ if (IS_ERR(*new_nsp)) {
```

```
   err = PTR_ERR(*new_nsp);
+  goto out;
+ }
+
+ err = ns_cgroup_clone(current);
+ if (err)
+  put_nsproxy(*new_nsp);
+
+out:
  return err;
 }


_

--

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers
```

Subject: [PATCH 17/29] containers implement namespace tracking subsystem fix order of container subsystems in

Posted by Paul Menage on Tue, 11 Sep 2007 19:52:56 GMT

View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>

some cosmetic changes to init/Kconfig to make the subsystem list
under cgroups look nicer when CONFIG_CGROUPS=y.

also changed the 'select' to 'depends on' for the namespace
subsystem.

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
Acked-by: Paul Menage <menage@google.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

 init/Kconfig | 32 ++++++++++++++++++++----------
 1 file changed, 16 insertions(+), 16 deletions(-)

diff -puN
init/Kconfig~cgroups-implement-namespace-tracking-subsystem-fix-order-of-cgroup-subsystems-in-init-kconfig init/Kconfig
---
a/init/Kconfig~cgroups-implement-namespace-tracking-subsystem-fix-order-of-cgroup-subsystems-in-init-kconfig

```
+++ a/init/Kconfig
@@ -292,6 +292,22 @@ config CGROUP_DEBUG

   Say N if unsure

+config CGROUP_NS
+       bool "Namespace cgroup subsystem"
+       depends on CGROUPS
+       help
+         Provides a simple namespace cgroup subsystem to
+         provide hierarchical naming of sets of namespaces,
+         for instance virtual servers and checkpoint/restart
+         jobs.
+
+config CGROUP_CPUACCT
+ bool "Simple CPU accounting cgroup subsystem"
+ depends on CGROUPS
+ help
+   Provides a simple Resource Controller for monitoring the
+   total CPU consumed by the tasks in a cgroup
+
 config CPUSETS
  bool "Cpuset support"
  depends on SMP && CGROUPS
@@ -323,27 +339,11 @@ config SYSFS_DEPRECATED
    If you are using a distro that was released in 2006 or later,
    it should be safe to say N here.

-config CGROUP_NS
-       bool "Namespace cgroup subsystem"
-       select CGROUPS
-       help
-         Provides a simple namespace cgroup subsystem to
-         provide hierarchical naming of sets of namespaces,
-         for instance virtual servers and checkpoint/restart
-         jobs.
-
 config PROC_PID_CPUSET
  bool "Include legacy /proc/<pid>/cpuset file"
  depends on CPUSETS
  default y

-config CGROUP_CPUACCT
- bool "Simple CPU accounting cgroup subsystem"
- depends on CGROUPS
- help
-   Provides a simple Resource Controller for monitoring the
-   total CPU consumed by the tasks in a cgroup
```

-
 config RELAY
  bool "Kernel->user space relay support (formerly relayfs)"
  help

_

--

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 18/29] memory controller add documentation
Posted by Paul Menage on Tue, 11 Sep 2007 19:52:57 GMT
View Forum Message <> Reply to Message

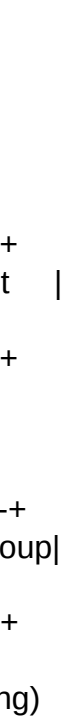
From: Balbir Singh <balbir@linux.vnet.ibm.com>

Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>
Cc: Pavel Emelianov <xemul@openvz.org>
Cc: Paul Menage <menage@google.com>
Cc: Peter Zijlstra <a.p.zijlstra@chello.nl>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Nick Piggin <nickpiggin@yahoo.com.au>
Cc: Kirill Korotaev <dev@sw.ru>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: David Rientjes <rientjes@google.com>
Cc: Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>

 Documentation/controllers/memory.txt | 259 ++++++++++++++++++++++++++++++++++
 1 files changed, 259 insertions(+)

diff -puN /dev/null Documentation/controllers/memory.txt
--- /dev/null
+++ a/Documentation/controllers/memory.txt
@@ -0,0 +1,259 @@
+Memory Controller
+
+Salient features
+
+a. Enable control of both RSS (mapped) and Page Cache (unmapped) pages
+b. The infrastructure allows easy addition of other types of memory to control
+c. Provides *zero overhead* for non memory controller users
+d. Provides a double LRU: global memory pressure causes reclaim from the
+   global LRU; a cgroup on hitting a limit, reclaims from the per

+   cgroup LRU
+
+NOTE: Page Cache (unmapped) also includes Swap Cache pages as a subset
+and will not be referred to explicitly in the rest of the documentation.
+
+Benefits and Purpose of the memory controller
+
+The memory controller isolates the memory behaviour of a group of tasks
+from the rest of the system. The article on LWN [12] mentions some probable
+uses of the memory controller. The memory controller can be used to
+
+a. Isolate an application or a group of applications
+   Memory hungry applications can be isolated and limited to a smaller
+   amount of memory.
+b. Create a cgroup with limited amount of memory, this can be used
+   as a good alternative to booting with mem=XXXX.
+c. Virtualization solutions can control the amount of memory they want
+   to assign to a virtual machine instance.
+d. A CD/DVD burner could control the amount of memory used by the
+   rest of the system to ensure that burning does not fail due to lack
+   of available memory.
+e. There are several other use cases, find one or use the controller just
+   for fun (to learn and hack on the VM subsystem).
+
+1. History
+
+The memory controller has a long history. A request for comments for the memory
+controller was posted by Balbir Singh [1]. At the time the RFC was posted
+there were several implementations for memory control. The goal of the
+RFC was to build consensus and agreement for the minimal features required
+for memory control. The first RSS controller was posted by Balbir Singh[2]
+in Feb 2007. Pavel Emelianov [3][4][5] has since posted three versions of the
+RSS controller. At OLS, at the resource management BoF, everyone suggested
+that we handle both page cache and RSS together. Another request was raised
+to allow user space handling of OOM. The current memory controller is
+at version 6; it combines both mapped (RSS) and unmapped Page
+Cache Control [11].
+
+2. Memory Control
+
+Memory is a unique resource in the sense that it is present in a limited
+amount. If a task requires a lot of CPU processing, the task can spread
+its processing over a period of hours, days, months or years, but with
+memory, the same physical memory needs to be reused to accomplish the task.
+
+The memory controller implementation has been divided into phases. These
+are:
+

+1. Memory controller
+2. mlock(2) controller
+3. Kernel user memory accounting and slab control
+4. user mappings length controller
+
+The memory controller is the first controller developed.
+
+2.1. Design
+
+The core of the design is a counter called the res_counter. The res_counter
+tracks the current memory usage and limit of the group of processes associated
+with the controller. Each cgroup has a memory controller specific data
+structure (mem_cgroup) associated with it.
+
+2.2. Accounting
+
+ 

+
+ (Figure 1: Hierarchy of Accounting)
+
+
+Figure 1 shows the important aspects of the controller
+
+1. Accounting happens per cgroup
+2. Each mm_struct knows about which cgroup it belongs to
+3. Each page has a pointer to the page_cgroup, which in turn knows the
+ cgroup it belongs to
+
+The accounting is done as follows: mem_cgroup_charge() is invoked to setup
+the necessary data structures and check if the cgroup that is being charged
+is over its limit. If it is then reclaim is invoked on the cgroup.
+More details can be found in the reclaim section of this document.

+If everything goes well, a page meta-data-structure called page_cgroup is
+allocated and associated with the page. This routine also adds the page to
+the per cgroup LRU.
+
+2.2.1 Accounting details
+
+All mapped pages (RSS) and unmapped user pages (Page Cache) are accounted.
+RSS pages are accounted at the time of page_add_*_rmap() unless they've already
+been accounted for earlier. A file page will be accounted for as Page Cache;
+it's mapped into the page tables of a process, duplicate accounting is carefully
+avoided. Page Cache pages are accounted at the time of add_to_page_cache().
+The corresponding routines that remove a page from the page tables or removes
+a page from Page Cache is used to decrement the accounting counters of the
+cgroup.
+
+2.3 Shared Page Accounting
+
+Shared pages are accounted on the basis of the first touch approach. The
+cgroup that first touches a page is accounted for the page. The principle
+behind this approach is that a cgroup that aggressively uses a shared
+page will eventually get charged for it (once it is uncharged from
+the cgroup that brought it in -- this will happen on memory pressure).
+
+2.4 Reclaim
+
+Each cgroup maintains a per cgroup LRU that consists of an active
+and inactive list. When a cgroup goes over its limit, we first try
+to reclaim memory from the cgroup so as to make space for the new
+pages that the cgroup has touched. If the reclaim is unsuccessful,
+an OOM routine is invoked to select and kill the bulkiest task in the
+cgroup.
+
+The reclaim algorithm has not been modified for cgroups, except that
+pages that are selected for reclaiming come from the per cgroup LRU
+list.
+
+2. Locking
+
+The memory controller uses the following hierarchy
+
+1. zone->lru_lock is used for selecting pages to be isolated
+2. mem->lru_lock protects the per cgroup LRU
+3. lock_page_cgroup() is used to protect page->page_cgroup
+
+3. User Interface
+
+0. Configuration
+

+a. Enable CONFIG_CGROUPS
+b. Enable CONFIG_RESOURCE_COUNTERS
+c. Enable CONFIG_CGROUP_MEM_CONT
+
+1. Prepare the cgroups
+# mkdir -p /cgroups
+# mount -t cgroup none /cgroups -o memory
+
+2. Make the new group and move bash into it
+# mkdir /cgroups/0
+# echo $$ > /cgroups/0/tasks
+
+Since now we're in the 0 cgroup,
+We can alter the memory limit:
+# echo -n 6000 > /cgroups/0/memory.limit
+
+We can check the usage:
+# cat /cgroups/0/memory.usage
+25
+
+The memory.failcnt field gives the number of times that the cgroup limit was
+exceeded.
+
+4. Testing
+
+Balbir posted lmbench, AIM9, LTP and vmmstress results [10] and [11].
+Apart from that v6 has been tested with several applications and regular
+daily use. The controller has also been tested on the PPC64, x86_64 and
+UML platforms.
+
+4.1 Troubleshooting
+
+Sometimes a user might find that the application under a cgroup is
+terminated. There are several causes for this:
+
+1. The cgroup limit is too low (just too low to do anything useful)
+2. The user is using anonymous memory and swap is turned off or too low
+
+A sync followed by echo 1 > /proc/sys/vm/drop_caches will help get rid of
+some of the pages cached in the cgroup (page cache pages).
+
+4.2 Task migration
+
+When a task migrates from one cgroup to another, it's charge is not
+carried forward. The pages allocated from the original cgroup still
+remain charged to it, the charge is dropped when the page is freed or
+reclaimed.
+

+4.3 Removing a cgroup
+
+A cgroup can be removed by rmdir, but as discussed in sections 4.1 and 4.2, a
+cgroup might have some charge associated with it, even though all
+tasks have migrated away from it. If some pages are still left, after following
+the steps listed in sections 4.1 and 4.2, check the Swap Cache usage in
+/proc/meminfo to see if the Swap Cache usage is showing up in the
+cgroups memory.usage counter. A simple test of swapoff -a and swapon -a
+should free any pending Swap Cache usage.
+
+4.4 Choosing what to account -- Page Cache (unmapped) vs RSS (mapped)?
+
+The type of memory accounted by the cgroup can be limited to just
+mapped pages by writing "1" to memory.control_type field
+
+echo -n 1 > memory.control_type
+
+5. TODO
+
+1. Add support for accounting huge pages (as a separate controller)
+2. Improve the user interface to accept/display memory limits in KB or MB
+   rather than pages (since page sizes can differ across platforms/machines).
+3. Make cgroup lists per-zone
+4. Make per-cgroup scanner reclaim not-shared pages first
+5. Teach controller to account for shared-pages
+6. Start reclamation when the limit is lowered
+7. Start reclamation in the background when the limit is
+   not yet hit but the usage is getting closer
+8. Create per zone LRU lists per cgroup
+
+Summary
+
+Overall, the memory controller has been a stable controller and has been
+commented and discussed quite extensively in the community.
+
+References
+
+1. Singh, Balbir. RFC: Memory Controller, http://lwn.net/Articles/206697/
+2. Singh, Balbir. Memory Controller (RSS Control),
+   http://lwn.net/Articles/222762/
+3. Emelianov, Pavel. Resource controllers based on process cgroups
+   http://lkml.org/lkml/2007/3/6/198
+4. Emelianov, Pavel. RSS controller based on process cgroups (v2)
+   http://lkml.org/lkml/2007/4/9/74
+5. Emelianov, Pavel. RSS controller based on process cgroups (v3)
+   http://lkml.org/lkml/2007/5/30/244
+6. Menage, Paul. Control Groups v10, http://lwn.net/Articles/236032/
+7. Vaidyanathan, Srinivasan, Control Groups: Pagecache accounting and control

+   subsystem (v3), http://lwn.net/Articles/235534/
+8. Singh, Balbir. RSS controller V2 test results (lmbench),
+   http://lkml.org/lkml/2007/5/17/232
+9. Singh, Balbir. RSS controller V2 AIM9 results
+   http://lkml.org/lkml/2007/5/18/1
+10. Singh, Balbir. Memory controller v6 results,
+   http://lkml.org/lkml/2007/8/19/36
+11. Singh, Balbir. Memory controller v6, http://lkml.org/lkml/2007/8/17/69
+12. Corbet, Jonathan, Controlling memory use in cgroups,
+   http://lwn.net/Articles/243795/

–

--

Subject: [PATCH 19/29] memory controller resource counters v7
Posted by Paul Menage on Tue, 11 Sep 2007 19:52:58 GMT

View Forum Message <> Reply to Message

From: Pavel Emelianov <xemul@openvz.org>

Introduce generic structures and routines for resource accounting.

Each resource accounting cgroup is supposed to aggregate it,
cgroup_subsystem_state and its resource-specific members within.

Signed-off-by: Pavel Emelianov <xemul@openvz.org>
Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>
Cc: Paul Menage <menage@google.com>
Cc: Peter Zijlstra <a.p.zijlstra@chello.nl>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Nick Piggin <nickpiggin@yahoo.com.au>
Cc: Kirill Korotaev <dev@sw.ru>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: David Rientjes <rientjes@google.com>
Cc: Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>

```
 include/linux/res_counter.h |  102 +++++++++++++++++++++++++++++
 init/Kconfig                |    7 +
 kernel/Makefile             |    1
 kernel/res_counter.c        |  120 ++++++++++++++++++++++++++++++++++
 4 files changed, 230 insertions(+)
```

```
diff -puN /dev/null include/linux/res_counter.h
--- /dev/null
+++ a/include/linux/res_counter.h
@@ -0,0 +1,102 @@
+#ifndef __RES_COUNTER_H__
+#define __RES_COUNTER_H__
+
+/*
+ * Resource Counters
+ * Contain common data types and routines for resource accounting
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ *
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ *
+ */
+
+#include <linux/cgroup.h>
+
+/*
+ * The core object. the cgroup that wishes to account for some
+ * resource may include this counter into its structures and use
+ * the helpers described beyond
+ */
+
+struct res_counter {
+ /*
+  * the current resource consumption level
+  */
+ unsigned long usage;
+ /*
+  * the limit that usage cannot exceed
+  */
+ unsigned long limit;
+ /*
+  * the number of unsuccessful attempts to consume the resource
+  */
+ unsigned long failcnt;
+ /*
+  * the lock to protect all of the above.
+  * the routines below consider this to be IRQ-safe
+  */
+ spinlock_t lock;
+};
+
+/*
+ * Helpers to interact with userspace
```

```
+ * res_counter_read/_write - put/get the specified fields from the
+ * res_counter struct to/from the user
+ *
+ * @counter:     the counter in question
+ * @member:  the field to work with (see RES_xxx below)
+ * @buf:     the buffer to opeate on,...
+ * @nbytes:  its size...
+ * @pos:     and the offset.
+ */
+
+ssize_t res_counter_read(struct res_counter *counter, int member,
+  const char __user *buf, size_t nbytes, loff_t *pos);
+ssize_t res_counter_write(struct res_counter *counter, int member,
+  const char __user *buf, size_t nbytes, loff_t *pos);
+
+/*
+ * the field descriptors. one for each member of res_counter
+ */
+
+enum {
+ RES_USAGE,
+ RES_LIMIT,
+ RES_FAILCNT,
+};
+
+/*
+ * helpers for accounting
+ */
+
+void res_counter_init(struct res_counter *counter);
+
+/*
+ * charge - try to consume more resource.
+ *
+ * @counter: the counter
+ * @val: the amount of the resource. each controller defines its own
+ *       units, e.g. numbers, bytes, Kbytes, etc
+ *
+ * returns 0 on success and <0 if the counter->usage will exceed the
+ * counter->limit _locked call expects the counter->lock to be taken
+ */
+
+int res_counter_charge_locked(struct res_counter *counter, unsigned long val);
+int res_counter_charge(struct res_counter *counter, unsigned long val);
+
+/*
+ * uncharge - tell that some portion of the resource is released
+ *
```

```
+ * @counter: the counter
+ * @val: the amount of the resource
+ *
+ * these calls check for usage underflow and show a warning on the console
+ * _locked call expects the counter->lock to be taken
+ */
+
+void res_counter_uncharge_locked(struct res_counter *counter, unsigned long val);
+void res_counter_uncharge(struct res_counter *counter, unsigned long val);
+
+#endif
diff -puN init/Kconfig~memory-controller-resource-counters-v7 init/Kconfig
--- a/init/Kconfig~memory-controller-resource-counters-v7
+++ a/init/Kconfig
@@ -319,6 +319,13 @@ config CPUSETS

    Say N if unsure.

+config RESOURCE_COUNTERS
+ bool "Resource counters"
+ help
+   This option enables controller independent resource accounting
+         infrastructure that works with cgroups
+ depends on CGROUPS
+
 config SYSFS_DEPRECATED
  bool "Create deprecated sysfs files"
  default y
diff -puN kernel/Makefile~memory-controller-resource-counters-v7 kernel/Makefile
--- a/kernel/Makefile~memory-controller-resource-counters-v7
+++ a/kernel/Makefile
@@ -59,6 +59,7 @@ obj-$(CONFIG_RELAY) += relay.o
 obj-$(CONFIG_SYSCTL) += utsname_sysctl.o
 obj-$(CONFIG_TASK_DELAY_ACCT) += delayacct.o
 obj-$(CONFIG_TASKSTATS) += taskstats.o tsacct.o
+obj-$(CONFIG_RESOURCE_COUNTERS) += res_counter.o

 ifneq ($(CONFIG_SCHED_NO_NO_OMIT_FRAME_POINTER),y)
 # According to Alan Modra <alan@linuxcare.com.au>, the -fno-omit-frame-pointer is
diff -puN /dev/null kernel/res_counter.c
--- /dev/null
+++ a/kernel/res_counter.c
@@ -0,0 +1,120 @@
+/*
+ * resource cgroups
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ *
```

```
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ *
+ */
+
+#include <linux/types.h>
+#include <linux/parser.h>
+#include <linux/fs.h>
+#include <linux/res_counter.h>
+#include <linux/uaccess.h>
+
+void res_counter_init(struct res_counter *counter)
+{
+ spin_lock_init(&counter->lock);
+ counter->limit = (unsigned long)LONG_MAX;
+}
+
+int res_counter_charge_locked(struct res_counter *counter, unsigned long val)
+{
+ if (counter->usage > (counter->limit - val)) {
+  counter->failcnt++;
+  return -ENOMEM;
+ }
+
+ counter->usage += val;
+ return 0;
+}
+
+int res_counter_charge(struct res_counter *counter, unsigned long val)
+{
+ int ret;
+ unsigned long flags;
+
+ spin_lock_irqsave(&counter->lock, flags);
+ ret = res_counter_charge_locked(counter, val);
+ spin_unlock_irqrestore(&counter->lock, flags);
+ return ret;
+}
+
+void res_counter_uncharge_locked(struct res_counter *counter, unsigned long val)
+{
+ if (WARN_ON(counter->usage < val))
+  val = counter->usage;
+
+ counter->usage -= val;
+}
+
+void res_counter_uncharge(struct res_counter *counter, unsigned long val)
+{
```

```
+ unsigned long flags;
+
+ spin_lock_irqsave(&counter->lock, flags);
+ res_counter_uncharge_locked(counter, val);
+ spin_unlock_irqrestore(&counter->lock, flags);
+}
+
+
+static inline unsigned long *res_counter_member(struct res_counter *counter,
+      int member)
+{
+ switch (member) {
+ case RES_USAGE:
+  return &counter->usage;
+ case RES_LIMIT:
+  return &counter->limit;
+ case RES_FAILCNT:
+  return &counter->failcnt;
+ };
+
+ BUG();
+ return NULL;
+}
+
+ssize_t res_counter_read(struct res_counter *counter, int member,
+  const char __user *userbuf, size_t nbytes, loff_t *pos)
+{
+ unsigned long *val;
+ char buf[64], *s;
+
+ s = buf;
+ val = res_counter_member(counter, member);
+ s += sprintf(s, "%lu\n", *val);
+ return simple_read_from_buffer((void __user *)userbuf, nbytes,
+   pos, buf, s - buf);
+}
+
+ssize_t res_counter_write(struct res_counter *counter, int member,
+  const char __user *userbuf, size_t nbytes, loff_t *pos)
+{
+ int ret;
+ char *buf, *end;
+ unsigned long tmp, *val;
+
+ buf = kmalloc(nbytes + 1, GFP_KERNEL);
+ ret = -ENOMEM;
+ if (buf == NULL)
+  goto out;
```

```
+
+ buf[nbytes] = '\0';
+ ret = -EFAULT;
+ if (copy_from_user(buf, userbuf, nbytes))
+  goto out_free;
+
+ ret = -EINVAL;
+ tmp = simple_strtoul(buf, &end, 10);
+ if (*end != '\0')
+  goto out_free;
+
+ val = res_counter_member(counter, member);
+ *val = tmp;
+ ret = nbytes;
+out_free:
+ kfree(buf);
+out:
+ return ret;
+}
```

–

--

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 20/29] memory controller resource counters v7 fix
Posted by Paul Menage on Tue, 11 Sep 2007 19:52:59 GMT
View Forum Message <> Reply to Message

From: David Rientjes <rientjes@google.com>

There's a gotcha in res_counter_charge_locked() because of C99 6.3.1.8(1) since both counter->limit and 'val' are of unsigned long type, the result of the subtraction will be the same; no promotion is required. So if 'val' is greater than counter->limit, it will always be larger than counter->usage and the conditional will fail. Simply casting this to signed doesn't work since counter->usage is also unsigned and thus the result of the subtraction will be promoted to unsigned since the ranks are the same.

Even though the only (current) use of res_counter_charge() is with a 'val' actual of 1, this still fails if you set counter->limit to 0. No chance of overflow unless you're running on a machine with 4KB pages and 16TB of memory.

Signed-off-by: David Rientjes <rientjes@google.com>
Cc: Pavel Emelianov <xemul@openvz.org>
Cc: Balbir Singh <balbir@linux.vnet.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

 kernel/res_counter.c | 2 +-
 1 files changed, 1 insertion(+), 1 deletion(-)

diff -puN kernel/res_counter.c~memory-controller-resource-counters-v7-fix kernel/res_counter.c
--- a/kernel/res_counter.c~memory-controller-resource-counters-v7-fix
+++ a/kernel/res_counter.c
@@ -21,7 +21,7 @@ void res_counter_init(struct res_counter

 int res_counter_charge_locked(struct res_counter *counter, unsigned long val)
 {
- if (counter->usage > (counter->limit - val)) {
+ if (counter->usage + val > counter->limit) {
   counter->failcnt++;
   return -ENOMEM;
  }
_

--

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 21/29] memory controller containers setup v7
Posted by Paul Menage on Tue, 11 Sep 2007 19:53:00 GMT
View Forum Message <> Reply to Message

From: Balbir Singh <balbir@linux.vnet.ibm.com>

Setup the memory cgroup and add basic hooks and controls to integrate
and work with the cgroup.

Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>
Cc: Pavel Emelianov <xemul@openvz.org>
Cc: Paul Menage <menage@google.com>
Cc: Peter Zijlstra <a.p.zijlstra@chello.nl>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Nick Piggin <nickpiggin@yahoo.com.au>
Cc: Kirill Korotaev <dev@sw.ru>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: David Rientjes <rientjes@google.com>

Cc: Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

 include/linux/cgroup_subsys.h | 6 +
 include/linux/memcontrol.h | 21 ++++
 init/Kconfig | 7 +
 mm/Makefile | 1
 mm/memcontrol.c | 127 ++
 5 files changed, 162 insertions(+)

diff -puN include/linux/cgroup_subsys.h~memory-controller-cgroups-setup-v7 include/linux/cgroup_subsys.h
--- a/include/linux/cgroup_subsys.h~memory-controller-cgroups-setup-v7
+++ a/include/linux/cgroup_subsys.h
@@ -30,3 +30,9 @@ SUBSYS(ns)
 #endif

 /* */
+
+#ifdef CONFIG_CGROUP_MEM_CONT
+SUBSYS(mem_cgroup)
+#endif
+
+/* */
diff -puN /dev/null include/linux/memcontrol.h
--- /dev/null
+++ a/include/linux/memcontrol.h
@@ -0,0 +1,21 @@
+/* memcontrol.h - Memory Controller
+ *
+ * Copyright IBM Corporation, 2007
+ * Author Balbir Singh <balbir@linux.vnet.ibm.com>
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+ * GNU General Public License for more details.
+ */
+
+#ifndef _LINUX_MEMCONTROL_H
+#define _LINUX_MEMCONTROL_H
+

```
+#endif /* _LINUX_MEMCONTROL_H */
+
diff -puN init/Kconfig~memory-controller-cgroups-setup-v7 init/Kconfig
--- a/init/Kconfig~memory-controller-cgroups-setup-v7
+++ a/init/Kconfig
@@ -346,6 +346,13 @@ config SYSFS_DEPRECATED
    If you are using a distro that was released in 2006 or later,
    it should be safe to say N here.

+config CGROUP_MEM_CONT
+ bool "Memory controller for cgroups"
+ depends on CGROUPS && RESOURCE_COUNTERS
+ help
+   Provides a memory controller that manages both page cache and
+   RSS memory.
+
 config PROC_PID_CPUSET
  bool "Include legacy /proc/<pid>/cpuset file"
  depends on CPUSETS
diff -puN mm/Makefile~memory-controller-cgroups-setup-v7 mm/Makefile
--- a/mm/Makefile~memory-controller-cgroups-setup-v7
+++ a/mm/Makefile
@@ -30,4 +30,5 @@ obj-$(CONFIG_FS_XIP) += filemap_xip.o
 obj-$(CONFIG_MIGRATION) += migrate.o
 obj-$(CONFIG_SMP) += allocpercpu.o
 obj-$(CONFIG_QUICKLIST) += quicklist.o
+obj-$(CONFIG_CGROUP_MEM_CONT) += memcontrol.o

diff -puN /dev/null mm/memcontrol.c
--- /dev/null
+++ a/mm/memcontrol.c
@@ -0,0 +1,127 @@
+/* memcontrol.c - Memory Controller
+ *
+ * Copyright IBM Corporation, 2007
+ * Author Balbir Singh <balbir@linux.vnet.ibm.com>
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
+ * GNU General Public License for more details.
+ */
+
```

```
+#include <linux/res_counter.h>
+#include <linux/memcontrol.h>
+#include <linux/cgroup.h>
+
+struct cgroup_subsys mem_cgroup_subsys;
+
+/*
+ * The memory controller data structure. The memory controller controls both
+ * page cache and RSS per cgroup. We would eventually like to provide
+ * statistics based on the statistics developed by Rik Van Riel for clock-pro,
+ * to help the administrator determine what knobs to tune.
+ *
+ * TODO: Add a water mark for the memory controller. Reclaim will begin when
+ * we hit the water mark.
+ */
+struct mem_cgroup {
+ struct cgroup_subsys_state css;
+ /*
+  * the counter to account for memory usage
+  */
+ struct res_counter res;
+};
+
+/*
+ * A page_cgroup page is associated with every page descriptor. The
+ * page_cgroup helps us identify information about the cgroup
+ */
+struct page_cgroup {
+ struct list_head lru;  /* per cgroup LRU list */
+ struct page *page;
+ struct mem_cgroup *mem_cgroup;
+};
+
+
+static inline
+struct mem_cgroup *mem_cgroup_from_cont(struct cgroup *cont)
+{
+ return container_of(cgroup_subsys_state(cont,
+    mem_cgroup_subsys_id), struct mem_cgroup,
+    css);
+}
+
+static ssize_t mem_cgroup_read(struct cgroup *cont, struct cftype *cft,
+   struct file *file, char __user *userbuf, size_t nbytes,
+   loff_t *ppos)
+{
+ return res_counter_read(&mem_cgroup_from_cont(cont)->res,
+    cft->private, userbuf, nbytes, ppos);
```

```
+}
+
+static ssize_t mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
+    struct file *file, const char __user *userbuf,
+    size_t nbytes, loff_t *ppos)
+{
+ return res_counter_write(&mem_cgroup_from_cont(cont)->res,
+    cft->private, userbuf, nbytes, ppos);
+}
+
+static struct cftype mem_cgroup_files[] = {
+ {
+  .name = "usage",
+  .private = RES_USAGE,
+  .read = mem_cgroup_read,
+ },
+ {
+  .name = "limit",
+  .private = RES_LIMIT,
+  .write = mem_cgroup_write,
+  .read = mem_cgroup_read,
+ },
+ {
+  .name = "failcnt",
+  .private = RES_FAILCNT,
+  .read = mem_cgroup_read,
+ },
+};
+
+static struct cgroup_subsys_state *
+mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
+{
+ struct mem_cgroup *mem;
+
+ mem = kzalloc(sizeof(struct mem_cgroup), GFP_KERNEL);
+ if (!mem)
+  return -ENOMEM;
+
+ res_counter_init(&mem->res);
+ return &mem->css;
+}
+
+static void mem_cgroup_destroy(struct cgroup_subsys *ss,
+    struct cgroup *cont)
+{
+ kfree(mem_cgroup_from_cont(cont));
+}
+
```

```
+static int mem_cgroup_populate(struct cgroup_subsys *ss,
+    struct cgroup *cont)
+{
+ return cgroup_add_files(cont, ss, mem_cgroup_files,
+     ARRAY_SIZE(mem_cgroup_files));
+}
+
+struct cgroup_subsys mem_cgroup_subsys = {
+ .name = "memory",
+ .subsys_id = mem_cgroup_subsys_id,
+ .create = mem_cgroup_create,
+ .destroy = mem_cgroup_destroy,
+ .populate = mem_cgroup_populate,
+ .early_init = 0,
+};
```

_

--

Subject: [PATCH 22/29] memory controller accounting setup v7
Posted by Paul Menage on Tue, 11 Sep 2007 19:53:01 GMT
View Forum Message <> Reply to Message

From: Pavel Emelianov <xemul@openvz.org>

Basic setup routines, the mm_struct has a pointer to the cgroup that
it belongs to and the the page has a page_cgroup associated with it.

Signed-off-by: Pavel Emelianov <xemul@openvz.org>
Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>
Cc: Paul Menage <menage@google.com>
Cc: Peter Zijlstra <a.p.zijlstra@chello.nl>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Nick Piggin <nickpiggin@yahoo.com.au>
Cc: Kirill Korotaev <dev@sw.ru>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: David Rientjes <rientjes@google.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>

```
 include/linux/memcontrol.h |   36 ++++++++++++++++++++++++++++++++++++
 include/linux/mm_types.h   |    6 +++
 include/linux/sched.h      |    1
```

```
 kernel/fork.c            |   11 ++++--
 mm/memcontrol.c          |   57 +++++++++++++++++++++++++++++++++++++---
 5 files changed, 104 insertions(+), 7 deletions(-)

diff -puN include/linux/memcontrol.h~memory-controller-accounting-setup-v7
include/linux/memcontrol.h
--- a/include/linux/memcontrol.h~memory-controller-accounting-setup-v7
+++ a/include/linux/memcontrol.h
@@ -3,6 +3,9 @@
  * Copyright IBM Corporation, 2007
  * Author Balbir Singh <balbir@linux.vnet.ibm.com>
  *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ *
  * This program is free software; you can redistribute it and/or modify
  * it under the terms of the GNU General Public License as published by
  * the Free Software Foundation; either version 2 of the License, or
@@ -17,5 +20,38 @@
 #ifndef _LINUX_MEMCONTROL_H
 #define _LINUX_MEMCONTROL_H

+struct mem_cgroup;
+struct page_cgroup;
+
+#ifdef CONFIG_CGROUP_MEM_CONT
+
+extern void mm_init_cgroup(struct mm_struct *mm, struct task_struct *p);
+extern void mm_free_cgroup(struct mm_struct *mm);
+extern void page_assign_page_cgroup(struct page *page,
+     struct page_cgroup *pc);
+extern struct page_cgroup *page_get_page_cgroup(struct page *page);
+
+#else /* CONFIG_CGROUP_MEM_CONT */
+static inline void mm_init_cgroup(struct mm_struct *mm,
+     struct task_struct *p)
+{
+}
+
+static inline void mm_free_cgroup(struct mm_struct *mm)
+{
+}
+
+static inline void page_assign_page_cgroup(struct page *page,
+      struct page_cgroup *pc)
+{
+}
+
```

```
+static inline struct page_cgroup *page_get_page_cgroup(struct page *page)
+{
+ return NULL;
+}
+
+#endif /* CONFIG_CGROUP_MEM_CONT */
+
 #endif /* _LINUX_MEMCONTROL_H */

diff -puN include/linux/mm_types.h~memory-controller-accounting-setup-v7
include/linux/mm_types.h
--- a/include/linux/mm_types.h~memory-controller-accounting-setup-v7
+++ a/include/linux/mm_types.h
@@ -83,6 +83,9 @@ struct page {
  void *virtual;   /* Kernel virtual address (NULL if
        not kmapped, ie. highmem) */
 #endif /* WANT_PAGE_VIRTUAL */
+#ifdef CONFIG_CGROUP_MEM_CONT
+ unsigned long page_cgroup;
+#endif
 };

 /*
@@ -214,6 +217,9 @@ struct mm_struct {
  /* aio bits */
  rwlock_t  ioctx_list_lock;
  struct kioctx  *ioctx_list;
+#ifdef CONFIG_CGROUP_MEM_CONT
+ struct mem_cgroup *mem_cgroup;
+#endif
 };

 #endif /* _LINUX_MM_TYPES_H */
diff -puN include/linux/sched.h~memory-controller-accounting-setup-v7 include/linux/sched.h
--- a/include/linux/sched.h~memory-controller-accounting-setup-v7
+++ a/include/linux/sched.h
@@ -88,6 +88,7 @@ struct sched_param {

 #include <asm/processor.h>

+struct mem_cgroup;
 struct exec_domain;
 struct futex_pi_state;
 struct bio;
diff -puN kernel/fork.c~memory-controller-accounting-setup-v7 kernel/fork.c
--- a/kernel/fork.c~memory-controller-accounting-setup-v7
+++ a/kernel/fork.c
@@ -51,6 +51,7 @@
```

```
 #include <linux/random.h>
 #include <linux/tty.h>
 #include <linux/proc_fs.h>
+#include <linux/memcontrol.h>

 #include <asm/pgtable.h>
 #include <asm/pgalloc.h>
@@ -329,7 +330,7 @@ __cacheline_aligned_in_smp DEFINE_SPINLO

 #include <linux/init_task.h>

-static struct mm_struct * mm_init(struct mm_struct * mm)
+static struct mm_struct * mm_init(struct mm_struct * mm, struct task_struct *p)
 {
  atomic_set(&mm->mm_users, 1);
  atomic_set(&mm->mm_count, 1);
@@ -346,11 +347,14 @@ static struct mm_struct * mm_init(struct
  mm->ioctx_list = NULL;
  mm->free_area_cache = TASK_UNMAPPED_BASE;
  mm->cached_hole_size = ~0UL;
+ mm_init_cgroup(mm, p);

  if (likely(!mm_alloc_pgd(mm))) {
   mm->def_flags = 0;
   return mm;
  }
+
+ mm_free_cgroup(mm);
  free_mm(mm);
  return NULL;
 }
@@ -365,7 +369,7 @@ struct mm_struct * mm_alloc(void)
  mm = allocate_mm();
  if (mm) {
   memset(mm, 0, sizeof(*mm));
-  mm = mm_init(mm);
+  mm = mm_init(mm, current);
  }
  return mm;
 }
@@ -379,6 +383,7 @@ void fastcall __mmdrop(struct mm_struct
 {
  BUG_ON(mm == &init_mm);
  mm_free_pgd(mm);
+ mm_free_cgroup(mm);
  destroy_context(mm);
  free_mm(mm);
 }
```

```
@@ -499,7 +504,7 @@ static struct mm_struct *dup_mm(struct t
  mm->token_priority = 0;
  mm->last_interval = 0;

- if (!mm_init(mm))
+ if (!mm_init(mm, tsk))
   goto fail_nomem;

  if (init_new_context(tsk, mm))
diff -puN mm/memcontrol.c~memory-controller-accounting-setup-v7 mm/memcontrol.c
--- a/mm/memcontrol.c~memory-controller-accounting-setup-v7
+++ a/mm/memcontrol.c
@@ -3,6 +3,9 @@
  * Copyright IBM Corporation, 2007
  * Author Balbir Singh <balbir@linux.vnet.ibm.com>
  *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ *
  * This program is free software; you can redistribute it and/or modify
  * it under the terms of the GNU General Public License as published by
  * the Free Software Foundation; either version 2 of the License, or
@@ -17,6 +20,7 @@
 #include <linux/res_counter.h>
 #include <linux/memcontrol.h>
 #include <linux/cgroup.h>
+#include <linux/mm.h>

 struct cgroup_subsys mem_cgroup_subsys;

@@ -35,6 +39,13 @@ struct mem_cgroup {
   * the counter to account for memory usage
   */
  struct res_counter res;
+ /*
+  * Per cgroup active and inactive list, similar to the
+  * per zone LRU lists.
+  * TODO: Consider making these lists per zone
+  */
+ struct list_head active_list;
+ struct list_head inactive_list;
 };

 /*
@@ -56,6 +67,37 @@ struct mem_cgroup *mem_cgroup_from
     css);
 }
```

```
+static inline
+struct mem_cgroup *mem_cgroup_from_task(struct task_struct *p)
+{
+ return container_of(task_subsys_state(p, mem_cgroup_subsys_id),
+     struct mem_cgroup, css);
+}
+
+void mm_init_cgroup(struct mm_struct *mm, struct task_struct *p)
+{
+ struct mem_cgroup *mem;
+
+ mem = mem_cgroup_from_task(p);
+ css_get(&mem->css);
+ mm->mem_cgroup = mem;
+}
+
+void mm_free_cgroup(struct mm_struct *mm)
+{
+ css_put(&mm->mem_cgroup->css);
+}
+
+void page_assign_page_cgroup(struct page *page, struct page_cgroup *pc)
+{
+ page->page_cgroup = (unsigned long)pc;
+}
+
+struct page_cgroup *page_get_page_cgroup(struct page *page)
+{
+ return page->page_cgroup;
+}
+
 static ssize_t mem_cgroup_read(struct cgroup *cont, struct cftype *cft,
     struct file *file, char __user *userbuf, size_t nbytes,
     loff_t *ppos)
@@ -91,14 +133,21 @@ static struct cftype mem_cgroup_files
  },
 };

+static struct mem_cgroup init_mem_cgroup;
+
 static struct cgroup_subsys_state *
 mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
 {
  struct mem_cgroup *mem;

- mem = kzalloc(sizeof(struct mem_cgroup), GFP_KERNEL);
- if (!mem)
-  return -ENOMEM;
```

```
+ if (unlikely((cont->parent) == NULL)) {
+  mem = &init_mem_cgroup;
+  init_mm.mem_cgroup = mem;
+ } else
+  mem = kzalloc(sizeof(struct mem_cgroup), GFP_KERNEL);
+
+ if (mem == NULL)
+  return NULL;

  res_counter_init(&mem->res);
  return &mem->css;
@@ -123,5 +172,5 @@ struct cgroup_subsys mem_cgroup_su
  .create = mem_cgroup_create,
  .destroy = mem_cgroup_destroy,
  .populate = mem_cgroup_populate,
- .early_init = 0,
+ .early_init = 1,
 };
_

--
```

Subject: [PATCH 23/29] memory controller memory accounting v7
Posted by Paul Menage on Tue, 11 Sep 2007 19:53:02 GMT
View Forum Message <> Reply to Message

From: Balbir Singh <balbir@linux.vnet.ibm.com>

Add the accounting hooks. The accounting is carried out for RSS and Page Cache (unmapped) pages. There is now a common limit and accounting for both. The RSS accounting is accounted at page_add_*_rmap() and page_remove_rmap() time. Page cache is accounted at add_to_page_cache(), __delete_from_page_cache(). Swap cache is also accounted for.

Each page's page_cgroup is protected with the last bit of the page_cgroup pointer, this makes handling of race conditions involving simultaneous mappings of a page easier. A reference count is kept in the page_cgroup to deal with cases where a page might be unmapped from the RSS of all tasks, but still lives in the page cache.

Credits go to Vaidyanathan Srinivasan for helping with reference counting work of the page cgroup. Almost all of the page cache accounting code has help from Vaidyanathan Srinivasan.

```
Signed-off-by: Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com>
Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>
Cc: Pavel Emelianov <xemul@openvz.org>
Cc: Paul Menage <menage@google.com>
Cc: Peter Zijlstra <a.p.zijlstra@chello.nl>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Nick Piggin <nickpiggin@yahoo.com.au>
Cc: Kirill Korotaev <dev@sw.ru>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: David Rientjes <rientjes@google.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

 include/linux/memcontrol.h |   20 ++++
 mm/filemap.c               |   12 ++
 mm/memcontrol.c            |  166 +++++++++++++++++++++++++++++++++++++++++-
 mm/memory.c                |   43 ++++++++-
 mm/migrate.c               |    6 +
 mm/page_alloc.c            |    3
 mm/rmap.c                  |   17 +++
 mm/swap_state.c            |   12 ++
 mm/swapfile.c              |   41 +++++---
 9 files changed, 293 insertions(+), 27 deletions(-)

diff -puN include/linux/memcontrol.h~memory-controller-memory-accounting-v7
include/linux/memcontrol.h
--- a/include/linux/memcontrol.h~memory-controller-memory-accounting-v7
+++ a/include/linux/memcontrol.h
@@ -30,6 +30,13 @@ extern void mm_free_cgroup(struct mm_
 extern void page_assign_page_cgroup(struct page *page,
      struct page_cgroup *pc);
 extern struct page_cgroup *page_get_page_cgroup(struct page *page);
+extern int mem_cgroup_charge(struct page *page, struct mm_struct *mm);
+extern void mem_cgroup_uncharge(struct page_cgroup *pc);
+
+static inline void mem_cgroup_uncharge_page(struct page *page)
+{
+ mem_cgroup_uncharge(page_get_page_cgroup(page));
+}

 #else /* CONFIG_CGROUP_MEM_CONT */
 static inline void mm_init_cgroup(struct mm_struct *mm,
@@ -51,6 +58,19 @@ static inline struct page_cgroup *pag
  return NULL;
 }

+static inline int mem_cgroup_charge(struct page *page, struct mm_struct *mm)
```

```
+{
+ return 0;
+}
+
+static inline void mem_cgroup_uncharge(struct page_cgroup *pc)
+{
+}
+
+static inline void mem_cgroup_uncharge_page(struct page *page)
+{
+}
+
 #endif /* CONFIG_CGROUP_MEM_CONT */

 #endif /* _LINUX_MEMCONTROL_H */
diff -puN mm/filemap.c~memory-controller-memory-accounting-v7 mm/filemap.c
--- a/mm/filemap.c~memory-controller-memory-accounting-v7
+++ a/mm/filemap.c
@@ -31,6 +31,7 @@
 #include <linux/syscalls.h>
 #include <linux/cpuset.h>
 #include <linux/hardirq.h> /* for BUG_ON(!in_atomic()) only */
+#include <linux/memcontrol.h>
 #include "internal.h"

 /*
@@ -116,6 +117,7 @@ void __remove_from_page_cache(struct pag
 {
  struct address_space *mapping = page->mapping;

+ mem_cgroup_uncharge_page(page);
  radix_tree_delete(&mapping->page_tree, page->index);
  page->mapping = NULL;
  mapping->nrpages--;
@@ -441,6 +443,11 @@ int add_to_page_cache(struct page *page,
  int error = radix_tree_preload(gfp_mask & ~__GFP_HIGHMEM);

  if (error == 0) {
+
+  error = mem_cgroup_charge(page, current->mm);
+  if (error)
+   goto out;
+
   write_lock_irq(&mapping->tree_lock);
   error = radix_tree_insert(&mapping->page_tree, offset, page);
   if (!error) {
@@ -450,10 +457,13 @@ int add_to_page_cache(struct page *page,
    page->index = offset;
```

```
    mapping->nrpages++;
    __inc_zone_page_state(page, NR_FILE_PAGES);
-  }
+  } else
+   mem_cgroup_uncharge_page(page);
+
   write_unlock_irq(&mapping->tree_lock);
   radix_tree_preload_end();
  }
+out:
  return error;
 }
 EXPORT_SYMBOL(add_to_page_cache);
diff -puN mm/memcontrol.c~memory-controller-memory-accounting-v7 mm/memcontrol.c
--- a/mm/memcontrol.c~memory-controller-memory-accounting-v7
+++ a/mm/memcontrol.c
@@ -21,6 +21,9 @@
 #include <linux/memcontrol.h>
 #include <linux/cgroup.h>
 #include <linux/mm.h>
+#include <linux/page-flags.h>
+#include <linux/bit_spinlock.h>
+#include <linux/rcupdate.h>

 struct cgroup_subsys mem_cgroup_subsys;

@@ -31,7 +34,9 @@ struct cgroup_subsys mem_cgroup_su
  * to help the administrator determine what knobs to tune.
  *
  * TODO: Add a water mark for the memory controller. Reclaim will begin when
- * we hit the water mark.
+ * we hit the water mark. May be even add a low water mark, such that
+ * no reclaim occurs from a cgroup at it's low water mark, this is
+ * a feature that will be implemented much later in the future.
  */
 struct mem_cgroup {
  struct cgroup_subsys_state css;
@@ -49,6 +54,14 @@ struct mem_cgroup {
 };

 /*
+ * We use the lower bit of the page->page_cgroup pointer as a bit spin
+ * lock. We need to ensure that page->page_cgroup is atleast two
+ * byte aligned (based on comments from Nick Piggin)
+ */
+#define PAGE_CGROUP_LOCK_BIT  0x0
+#define PAGE_CGROUP_LOCK   (1 << PAGE_CGROUP_LOCK_BIT)
+
```

```
+/*
  * A page_cgroup page is associated with every page descriptor. The
  * page_cgroup helps us identify information about the cgroup
  */
@@ -56,6 +69,8 @@ struct page_cgroup {
  struct list_head lru;  /* per cgroup LRU list */
  struct page *page;
  struct mem_cgroup *mem_cgroup;
+ atomic_t ref_cnt;  /* Helpful when pages move b/w  */
+     /* mapped and cached states     */
 };


@@ -88,14 +103,157 @@ void mm_free_cgroup(struct mm_struct
  css_put(&mm->mem_cgroup->css);
 }

+static inline int page_cgroup_locked(struct page *page)
+{
+ return bit_spin_is_locked(PAGE_CGROUP_LOCK_BIT,
+     &page->page_cgroup);
+}
+
 void page_assign_page_cgroup(struct page *page, struct page_cgroup *pc)
 {
- page->page_cgroup = (unsigned long)pc;
+ int locked;
+
+ /*
+  * While resetting the page_cgroup we might not hold the
+  * page_cgroup lock. free_hot_cold_page() is an example
+  * of such a scenario
+  */
+ if (pc)
+  VM_BUG_ON(!page_cgroup_locked(page));
+ locked = (page->page_cgroup & PAGE_CGROUP_LOCK);
+ page->page_cgroup = ((unsigned long)pc | locked);
 }

 struct page_cgroup *page_get_page_cgroup(struct page *page)
 {
- return page->page_cgroup;
+ return (struct page_cgroup *)
+  (page->page_cgroup & ~PAGE_CGROUP_LOCK);
+}
+
+void __always_inline lock_page_cgroup(struct page *page)
+{
```

```
+ bit_spin_lock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
+ VM_BUG_ON(!page_cgroup_locked(page));
+}
+
+void __always_inline unlock_page_cgroup(struct page *page)
+{
+ bit_spin_unlock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
+}
+
+/*
+ * Charge the memory controller for page usage.
+ * Return
+ * 0 if the charge was successful
+ * < 0 if the cgroup is over its limit
+ */
+int mem_cgroup_charge(struct page *page, struct mm_struct *mm)
+{
+ struct mem_cgroup *mem;
+ struct page_cgroup *pc, *race_pc;
+
+ /*
+  * Should page_cgroup's go to their own slab?
+  * One could optimize the performance of the charging routine
+  * by saving a bit in the page_flags and using it as a lock
+  * to see if the cgroup page already has a page_cgroup associated
+  * with it
+  */
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ /*
+  * The page_cgroup exists and the page has already been accounted
+  */
+ if (pc) {
+  atomic_inc(&pc->ref_cnt);
+  goto done;
+ }
+
+ unlock_page_cgroup(page);
+
+ pc = kzalloc(sizeof(struct page_cgroup), GFP_KERNEL);
+ if (pc == NULL)
+  goto err;
+
+ rcu_read_lock();
+ /*
+  * We always charge the cgroup the mm_struct belongs to
+  * the mm_struct's mem_cgroup changes on task migration if the
+  * thread group leader migrates. It's possible that mm is not
```

```
+  * set, if so charge the init_mm (happens for pagecache usage).
+  */
+ if (!mm)
+  mm = &init_mm;
+
+ mem = rcu_dereference(mm->mem_cgroup);
+ /*
+  * For every charge from the cgroup, increment reference
+  * count
+  */
+ css_get(&mem->css);
+ rcu_read_unlock();
+
+ /*
+  * If we created the page_cgroup, we should free it on exceeding
+  * the cgroup limit.
+  */
+ if (res_counter_charge(&mem->res, 1)) {
+  css_put(&mem->css);
+  goto free_pc;
+ }
+
+ lock_page_cgroup(page);
+ /*
+  * Check if somebody else beat us to allocating the page_cgroup
+  */
+ race_pc = page_get_page_cgroup(page);
+ if (race_pc) {
+  kfree(pc);
+  pc = race_pc;
+  atomic_inc(&pc->ref_cnt);
+  res_counter_uncharge(&mem->res, 1);
+  css_put(&mem->css);
+  goto done;
+ }
+
+ atomic_set(&pc->ref_cnt, 1);
+ pc->mem_cgroup = mem;
+ pc->page = page;
+ page_assign_page_cgroup(page, pc);
+
+done:
+ unlock_page_cgroup(page);
+ return 0;
+free_pc:
+ kfree(pc);
+ return -ENOMEM;
+err:
```

```
+ unlock_page_cgroup(page);
+ return -ENOMEM;
+}
+
+/*
+ * Uncharging is always a welcome operation, we never complain, simply
+ * uncharge.
+ */
+void mem_cgroup_uncharge(struct page_cgroup *pc)
+{
+ struct mem_cgroup *mem;
+ struct page *page;
+
+ if (!pc)
+  return;
+
+ if (atomic_dec_and_test(&pc->ref_cnt)) {
+  page = pc->page;
+  lock_page_cgroup(page);
+  mem = pc->mem_cgroup;
+  css_put(&mem->css);
+  page_assign_page_cgroup(page, NULL);
+  unlock_page_cgroup(page);
+  res_counter_uncharge(&mem->res, 1);
+  kfree(pc);
+ }
 }

 static ssize_t mem_cgroup_read(struct cgroup *cont, struct cftype *cft,
@@ -150,6 +308,8 @@ mem_cgroup_create(struct cgroup_su
   return NULL;

  res_counter_init(&mem->res);
+ INIT_LIST_HEAD(&mem->active_list);
+ INIT_LIST_HEAD(&mem->inactive_list);
  return &mem->css;
 }

diff -puN mm/memory.c~memory-controller-memory-accounting-v7 mm/memory.c
--- a/mm/memory.c~memory-controller-memory-accounting-v7
+++ a/mm/memory.c
@@ -50,6 +50,7 @@
 #include <linux/delayacct.h>
 #include <linux/init.h>
 #include <linux/writeback.h>
+#include <linux/memcontrol.h>

 #include <asm/pgalloc.h>
```

```
 #include <asm/uaccess.h>
@@ -1136,14 +1137,18 @@ static int insert_page(struct mm_struct
  pte_t *pte;
  spinlock_t *ptl;

+ retval = mem_cgroup_charge(page, mm);
+ if (retval)
+  goto out;
+
  retval = -EINVAL;
  if (PageAnon(page))
-  goto out;
+  goto out_uncharge;
  retval = -ENOMEM;
  flush_dcache_page(page);
  pte = get_locked_pte(mm, addr, &ptl);
  if (!pte)
-  goto out;
+  goto out_uncharge;
  retval = -EBUSY;
  if (!pte_none(*pte))
   goto out_unlock;
@@ -1155,8 +1160,11 @@ static int insert_page(struct mm_struct
  set_pte_at(mm, addr, pte, mk_pte(page, prot));

  retval = 0;
+ return retval;
 out_unlock:
  pte_unmap_unlock(pte, ptl);
+out_uncharge:
+ mem_cgroup_uncharge_page(page);
 out:
  return retval;
 }
@@ -1629,6 +1637,9 @@ gotten:
   goto oom;
  cow_user_page(new_page, old_page, address, vma);

+ if (mem_cgroup_charge(new_page, mm))
+  goto oom_free_new;
+
  /*
   * Re-check the pte - we dropped the lock
   */
@@ -1660,7 +1671,9 @@ gotten:
   /* Free the old page.. */
   new_page = old_page;
   ret |= VM_FAULT_WRITE;
```

```
- }
+ } else
+  mem_cgroup_uncharge_page(new_page);
+
  if (new_page)
   page_cache_release(new_page);
  if (old_page)
@@ -1681,6 +1694,8 @@ unlock:
   put_page(dirty_page);
  }
  return ret;
+oom_free_new:
+ __free_page(new_page);
 oom:
  if (old_page)
   page_cache_release(old_page);
@@ -2085,6 +2100,11 @@ static int do_swap_page(struct mm_struct
  }

  delayacct_clear_flag(DELAYACCT_PF_SWAPIN);
+ if (mem_cgroup_charge(page, mm)) {
+  ret = VM_FAULT_OOM;
+  goto out;
+ }
+
  mark_page_accessed(page);
  lock_page(page);

@@ -2121,8 +2141,10 @@ static int do_swap_page(struct mm_struct
  if (write_access) {
   /* XXX: We could OR the do_wp_page code with this one? */
   if (do_wp_page(mm, vma, address,
-    page_table, pmd, ptl, pte) & VM_FAULT_OOM)
+    page_table, pmd, ptl, pte) & VM_FAULT_OOM) {
+   mem_cgroup_uncharge_page(page);
    ret = VM_FAULT_OOM;
+  }
   goto out;
  }

@@ -2133,6 +2155,7 @@ unlock:
 out:
  return ret;
 out_nomap:
+ mem_cgroup_uncharge_page(page);
  pte_unmap_unlock(page_table, ptl);
  unlock_page(page);
  page_cache_release(page);
```

```
@@ -2161,6 +2184,9 @@ static int do_anonymous_page(struct mm_s
  if (!page)
   goto oom;

+  if (mem_cgroup_charge(page, mm))
+   goto oom_free_page;
+
  entry = mk_pte(page, vma->vm_page_prot);
  entry = maybe_mkwrite(pte_mkdirty(entry), vma);

@@ -2178,8 +2204,11 @@ unlock:
  pte_unmap_unlock(page_table, ptl);
  return 0;
 release:
+ mem_cgroup_uncharge_page(page);
  page_cache_release(page);
  goto unlock;
+oom_free_page:
+ __free_page(page);
 oom:
  return VM_FAULT_OOM;
 }
@@ -2290,6 +2319,11 @@ static int __do_fault(struct mm_struct *

  }

+ if (mem_cgroup_charge(page, mm)) {
+  ret = VM_FAULT_OOM;
+  goto out;
+ }
+
  page_table = pte_offset_map_lock(mm, pmd, address, &ptl);

  /*
@@ -2325,6 +2359,7 @@ static int __do_fault(struct mm_struct *
   /* no need to invalidate: a not-present page won't be cached */
   update_mmu_cache(vma, address, entry);
  } else {
+  mem_cgroup_uncharge_page(page);
   if (anon)
    page_cache_release(page);
   else
diff -puN mm/migrate.c~memory-controller-memory-accounting-v7 mm/migrate.c
--- a/mm/migrate.c~memory-controller-memory-accounting-v7
+++ a/mm/migrate.c
@@ -29,6 +29,7 @@
 #include <linux/mempolicy.h>
 #include <linux/vmalloc.h>
```

```
 #include <linux/security.h>
+#include <linux/memcontrol.h>

 #include "internal.h"

@@ -157,6 +158,11 @@ static void remove_migration_pte(struct
    return;
   }

+ if (mem_cgroup_charge(new, mm)) {
+  pte_unmap(ptep);
+  return;
+ }
+
   ptl = pte_lockptr(mm, pmd);
   spin_lock(ptl);
  pte = *ptep;
diff -puN mm/page_alloc.c~memory-controller-memory-accounting-v7 mm/page_alloc.c
--- a/mm/page_alloc.c~memory-controller-memory-accounting-v7
+++ a/mm/page_alloc.c
@@ -42,6 +42,7 @@
 #include <linux/backing-dev.h>
 #include <linux/fault-inject.h>
 #include <linux/page-isolation.h>
+#include <linux/memcontrol.h>

 #include <asm/tlbflush.h>
 #include <asm/div64.h>
@@ -1019,6 +1020,7 @@ static void fastcall free_hot_cold_page(

  if (!PageHighMem(page))
   debug_check_no_locks_freed(page_address(page), PAGE_SIZE);
+ page_assign_page_cgroup(page, NULL);
  arch_free_page(page, 0);
  kernel_map_pages(page, 1, 0);

@@ -2561,6 +2563,7 @@ void __meminit memmap_init_zone(unsigned
   set_page_links(page, zone, nid, pfn);
   init_page_count(page);
   reset_page_mapcount(page);
+  page_assign_page_cgroup(page, NULL);
   SetPageReserved(page);

   /*
diff -puN mm/rmap.c~memory-controller-memory-accounting-v7 mm/rmap.c
--- a/mm/rmap.c~memory-controller-memory-accounting-v7
+++ a/mm/rmap.c
@@ -48,6 +48,7 @@
```

```
 #include <linux/rcupdate.h>
 #include <linux/module.h>
 #include <linux/kallsyms.h>
+#include <linux/memcontrol.h>

 #include <asm/tlbflush.h>

@@ -550,8 +551,14 @@ void page_add_anon_rmap(struct page *pag
  VM_BUG_ON(address < vma->vm_start || address >= vma->vm_end);
  if (atomic_inc_and_test(&page->_mapcount))
   __page_set_anon_rmap(page, vma, address);
- else
+ else {
   __page_check_anon_rmap(page, vma, address);
+  /*
+   * We unconditionally charged during prepare, we uncharge here
+   * This takes care of balancing the reference counts
+   */
+  mem_cgroup_uncharge_page(page);
+ }
 }

 /*
@@ -582,6 +589,12 @@ void page_add_file_rmap(struct page *pag
 {
  if (atomic_inc_and_test(&page->_mapcount))
   __inc_zone_page_state(page, NR_FILE_MAPPED);
+ else
+  /*
+   * We unconditionally charged during prepare, we uncharge here
+   * This takes care of balancing the reference counts
+   */
+  mem_cgroup_uncharge_page(page);
 }

 #ifdef CONFIG_DEBUG_VM
@@ -642,6 +655,8 @@ void page_remove_rmap(struct page *page,
    page_clear_dirty(page);
    set_page_dirty(page);
   }
+  mem_cgroup_uncharge_page(page);
+
   __dec_zone_page_state(page,
    PageAnon(page) ? NR_ANON_PAGES : NR_FILE_MAPPED);
 }
diff -puN mm/swap_state.c~memory-controller-memory-accounting-v7 mm/swap_state.c
--- a/mm/swap_state.c~memory-controller-memory-accounting-v7
+++ a/mm/swap_state.c
```

```
@@ -16,6 +16,7 @@
 #include <linux/backing-dev.h>
 #include <linux/pagevec.h>
 #include <linux/migrate.h>
+#include <linux/memcontrol.h>

 #include <asm/pgtable.h>

@@ -79,6 +80,11 @@ static int __add_to_swap_cache(struct pa
  BUG_ON(PagePrivate(page));
  error = radix_tree_preload(gfp_mask);
  if (!error) {
+
+  error = mem_cgroup_charge(page, current->mm);
+  if (error)
+   goto out;
+
   write_lock_irq(&swapper_space.tree_lock);
   error = radix_tree_insert(&swapper_space.page_tree,
       entry.val, page);
@@ -88,10 +94,13 @@ static int __add_to_swap_cache(struct pa
    set_page_private(page, entry.val);
    total_swapcache_pages++;
    __inc_zone_page_state(page, NR_FILE_PAGES);
-  }
+  } else
+   mem_cgroup_uncharge_page(page);
+
   write_unlock_irq(&swapper_space.tree_lock);
   radix_tree_preload_end();
  }
+out:
  return error;
 }

@@ -131,6 +140,7 @@ void __delete_from_swap_cache(struct pag
  BUG_ON(PageWriteback(page));
  BUG_ON(PagePrivate(page));

+ mem_cgroup_uncharge_page(page);
  radix_tree_delete(&swapper_space.page_tree, page_private(page));
  set_page_private(page, 0);
  ClearPageSwapCache(page);
diff -puN mm/swapfile.c~memory-controller-memory-accounting-v7 mm/swapfile.c
--- a/mm/swapfile.c~memory-controller-memory-accounting-v7
+++ a/mm/swapfile.c
@@ -27,6 +27,7 @@
 #include <linux/mutex.h>
```

```
 #include <linux/capability.h>
 #include <linux/syscalls.h>
+#include <linux/memcontrol.h>

 #include <asm/pgtable.h>
 #include <asm/tlbflush.h>
@@ -506,9 +507,12 @@ unsigned int count_swap_pages(int type,
  * just let do_wp_page work it out if a write is requested later - to
  * force COW, vm_page_prot omits write permission from any private vma.
  */
-static void unuse_pte(struct vm_area_struct *vma, pte_t *pte,
+static int unuse_pte(struct vm_area_struct *vma, pte_t *pte,
   unsigned long addr, swp_entry_t entry, struct page *page)
 {
+ if (mem_cgroup_charge(page, vma->vm_mm))
+  return -ENOMEM;
+
  inc_mm_counter(vma->vm_mm, anon_rss);
  get_page(page);
  set_pte_at(vma->vm_mm, addr, pte,
@@ -520,6 +524,7 @@ static void unuse_pte(struct vm_area_str
   * immediately swapped out again after swapon.
   */
  activate_page(page);
+ return 1;
 }

 static int unuse_pte_range(struct vm_area_struct *vma, pmd_t *pmd,
@@ -529,7 +534,7 @@ static int unuse_pte_range(struct vm_are
  pte_t swp_pte = swp_entry_to_pte(entry);
  pte_t *pte;
  spinlock_t *ptl;
- int found = 0;
+ int ret = 0;

  pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
  do {
@@ -538,13 +543,12 @@ static int unuse_pte_range(struct vm_are
    * Test inline before going to call unuse_pte.
    */
   if (unlikely(pte_same(*pte, swp_pte))) {
-   unuse_pte(vma, pte++, addr, entry, page);
-   found = 1;
+   ret = unuse_pte(vma, pte++, addr, entry, page);
    break;
   }
  } while (pte++, addr += PAGE_SIZE, addr != end);
  pte_unmap_unlock(pte - 1, ptl);
```

```
- return found;
+ return ret;
 }

 static inline int unuse_pmd_range(struct vm_area_struct *vma, pud_t *pud,
@@ -553,14 +557,16 @@ static inline int unuse_pmd_range(struct
 {
  pmd_t *pmd;
  unsigned long next;
+ int ret;

  pmd = pmd_offset(pud, addr);
  do {
   next = pmd_addr_end(addr, end);
   if (pmd_none_or_clear_bad(pmd))
    continue;
-  if (unuse_pte_range(vma, pmd, addr, next, entry, page))
-   return 1;
+  ret = unuse_pte_range(vma, pmd, addr, next, entry, page);
+  if (ret)
+   return ret;
  } while (pmd++, addr = next, addr != end);
  return 0;
 }
@@ -571,14 +577,16 @@ static inline int unuse_pud_range(struct
 {
  pud_t *pud;
  unsigned long next;
+ int ret;

  pud = pud_offset(pgd, addr);
  do {
   next = pud_addr_end(addr, end);
   if (pud_none_or_clear_bad(pud))
    continue;
-  if (unuse_pmd_range(vma, pud, addr, next, entry, page))
-   return 1;
+  ret = unuse_pmd_range(vma, pud, addr, next, entry, page);
+  if (ret)
+   return ret;
  } while (pud++, addr = next, addr != end);
  return 0;
 }
@@ -588,6 +596,7 @@ static int unuse_vma(struct vm_area_stru
 {
  pgd_t *pgd;
  unsigned long addr, end, next;
+ int ret;
```

```
  if (page->mapping) {
   addr = page_address_in_vma(page, vma);
@@ -605,8 +614,9 @@ static int unuse_vma(struct vm_area_stru
   next = pgd_addr_end(addr, end);
   if (pgd_none_or_clear_bad(pgd))
    continue;
-  if (unuse_pud_range(vma, pgd, addr, next, entry, page))
-   return 1;
+  ret = unuse_pud_range(vma, pgd, addr, next, entry, page);
+  if (ret)
+   return ret;
  } while (pgd++, addr = next, addr != end);
  return 0;
 }
@@ -615,6 +625,7 @@ static int unuse_mm(struct mm_struct *mm
     swp_entry_t entry, struct page *page)
 {
  struct vm_area_struct *vma;
+ int ret = 0;

  if (!down_read_trylock(&mm->mmap_sem)) {
   /*
@@ -627,15 +638,11 @@ static int unuse_mm(struct mm_struct *mm
   lock_page(page);
  }
  for (vma = mm->mmap; vma; vma = vma->vm_next) {
-  if (vma->anon_vma && unuse_vma(vma, entry, page))
+  if (vma->anon_vma && (ret = unuse_vma(vma, entry, page)))
    break;
  }
  up_read(&mm->mmap_sem);
- /*
-  * Currently unuse_mm cannot fail, but leave error handling
-  * at call sites for now, since we change it from time to time.
-  */
- return 0;
+ return ret;
 }

 /*

--
```

Subject: [PATCH 24/29] memory controller task migration v7
Posted by Paul Menage on Tue, 11 Sep 2007 19:53:03 GMT
View Forum Message <> Reply to Message

From: Balbir Singh <balbir@linux.vnet.ibm.com>

Allow tasks to migrate from one cgroup to the other. We migrate
mm_struct's mem_cgroup only when the thread group id migrates.

Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>
Cc: Pavel Emelianov <xemul@openvz.org>
Cc: Paul Menage <menage@google.com>
Cc: Peter Zijlstra <a.p.zijlstra@chello.nl>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Nick Piggin <nickpiggin@yahoo.com.au>
Cc: Kirill Korotaev <dev@sw.ru>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: David Rientjes <rientjes@google.com>
Cc: Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

 mm/memcontrol.c | 35 +++++++++++++++++++++++++++++++++++
 1 files changed, 35 insertions(+)

diff -puN mm/memcontrol.c~memory-controller-task-migration-v7 mm/memcontrol.c
--- a/mm/memcontrol.c~memory-controller-task-migration-v7
+++ a/mm/memcontrol.c
@@ -326,11 +326,46 @@ static int mem_cgroup_populate(struct
     ARRAY_SIZE(mem_cgroup_files));
 }

+static void mem_cgroup_move_task(struct cgroup_subsys *ss,
+   struct cgroup *cont,
+   struct cgroup *old_cont,
+   struct task_struct *p)
+{
+ struct mm_struct *mm;
+ struct mem_cgroup *mem, *old_mem;
+
+ mm = get_task_mm(p);
+ if (mm == NULL)
+ return;
+
+ mem = mem_cgroup_from_cont(cont);
+ old_mem = mem_cgroup_from_cont(old_cont);
+
+ if (mem == old_mem)
+ goto out;

```
+
+ /*
+  * Only thread group leaders are allowed to migrate, the mm_struct is
+  * in effect owned by the leader
+  */
+ if (p->tgid != p->pid)
+  goto out;
+
+ css_get(&mem->css);
+ rcu_assign_pointer(mm->mem_cgroup, mem);
+ css_put(&old_mem->css);
+
+out:
+ mmput(mm);
+ return;
+}
+
 struct cgroup_subsys mem_cgroup_subsys = {
  .name = "memory",
  .subsys_id = mem_cgroup_subsys_id,
  .create = mem_cgroup_create,
  .destroy = mem_cgroup_destroy,
  .populate = mem_cgroup_populate,
+ .attach = mem_cgroup_move_task,
  .early_init = 1,
 };
_

--

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers
```

Subject: [PATCH 25/29] memory controller add per container lru and reclaim v7
Posted by Paul Menage on Tue, 11 Sep 2007 19:53:04 GMT
View Forum Message <> Reply to Message

From: Balbir Singh <balbir@linux.vnet.ibm.com>

Add the page_cgroup to the per cgroup LRU. The reclaim algorithm has
been modified to make the isolate_lru_pages() as a pluggable component. The
scan_control data structure now accepts the cgroup on behalf of which
reclaims are carried out. try_to_free_pages() has been extended to become
cgroup aware.

Signed-off-by: Pavel Emelianov <xemul@openvz.org>

```
Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>
Cc: Paul Menage <menage@google.com>
Cc: Peter Zijlstra <a.p.zijlstra@chello.nl>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Nick Piggin <nickpiggin@yahoo.com.au>
Cc: Kirill Korotaev <dev@sw.ru>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: David Rientjes <rientjes@google.com>
Cc: Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

 include/linux/memcontrol.h  |   12 +++
 include/linux/res_counter.h |   23 +++++
 include/linux/swap.h        |    3
 mm/memcontrol.c             |  135 ++++++++++++++++++++++++++++++++++++-
 mm/swap.c                   |    2
 mm/vmscan.c                 |  126 ++++++++++++++++++++++++++++------
 6 files changed, 275 insertions(+), 26 deletions(-)

diff -puN include/linux/memcontrol.h~memory-controller-add-per-cgroup-lru-and-reclaim-v7
include/linux/memcontrol.h
--- a/include/linux/memcontrol.h~memory-controller-add-per-cgroup-lru-and-reclaim-v7
+++ a/include/linux/memcontrol.h
@@ -32,6 +32,13 @@ extern void page_assign_page_cgroup(s
 extern struct page_cgroup *page_get_page_cgroup(struct page *page);
 extern int mem_cgroup_charge(struct page *page, struct mm_struct *mm);
 extern void mem_cgroup_uncharge(struct page_cgroup *pc);
+extern void mem_cgroup_move_lists(struct page_cgroup *pc, bool active);
+extern unsigned long mem_cgroup_isolate_pages(unsigned long nr_to_scan,
+	struct list_head *dst,
+	unsigned long *scanned, int order,
+	int mode, struct zone *z,
+	struct mem_cgroup *mem_cont,
+	int active);

 static inline void mem_cgroup_uncharge_page(struct page *page)
 {
@@ -71,6 +78,11 @@ static inline void mem_cgroup_uncharg
 {
 }

+static inline void mem_cgroup_move_lists(struct page_cgroup *pc,
+	bool active)
+{}
+
 #endif /* CONFIG_CGROUP_MEM_CONT */
```

```
 #endif /* _LINUX_MEMCONTROL_H */
diff -puN include/linux/res_counter.h~memory-controller-add-per-cgroup-lru-and-reclaim-v7
include/linux/res_counter.h
--- a/include/linux/res_counter.h~memory-controller-add-per-cgroup-lru-and-reclaim-v7
+++ a/include/linux/res_counter.h
@@ -99,4 +99,27 @@ int res_counter_charge(struct res_counte
 void res_counter_uncharge_locked(struct res_counter *counter, unsigned long val);
 void res_counter_uncharge(struct res_counter *counter, unsigned long val);

+static inline bool res_counter_limit_check_locked(struct res_counter *cnt)
+{
+ if (cnt->usage < cnt->limit)
+  return true;
+
+ return false;
+}
+
+/*
+ * Helper function to detect if the cgroup is within it's limit or
+ * not. It's currently called from cgroup_rss_prepare()
+ */
+static inline bool res_counter_check_under_limit(struct res_counter *cnt)
+{
+ bool ret;
+ unsigned long flags;
+
+ spin_lock_irqsave(&cnt->lock, flags);
+ ret = res_counter_limit_check_locked(cnt);
+ spin_unlock_irqrestore(&cnt->lock, flags);
+ return ret;
+}
+
 #endif
diff -puN include/linux/swap.h~memory-controller-add-per-cgroup-lru-and-reclaim-v7
include/linux/swap.h
--- a/include/linux/swap.h~memory-controller-add-per-cgroup-lru-and-reclaim-v7
+++ a/include/linux/swap.h
@@ -6,6 +6,7 @@
 #include <linux/mmzone.h>
 #include <linux/list.h>
 #include <linux/sched.h>
+#include <linux/memcontrol.h>

 #include <asm/atomic.h>
 #include <asm/page.h>
@@ -190,6 +191,8 @@ extern void swap_setup(void);
 /* linux/mm/vmscan.c */
```

```
 extern unsigned long try_to_free_pages(struct zone **zones, int order,
      gfp_t gfp_mask);
+extern unsigned long try_to_free_mem_cgroup_pages(struct mem_cgroup *mem);
+extern int __isolate_lru_page(struct page *page, int mode);
 extern unsigned long shrink_all_memory(unsigned long nr_pages);
 extern int vm_swappiness;
 extern int remove_mapping(struct address_space *mapping, struct page *page);
diff -puN mm/memcontrol.c~memory-controller-add-per-cgroup-lru-and-reclaim-v7
mm/memcontrol.c
--- a/mm/memcontrol.c~memory-controller-add-per-cgroup-lru-and-reclaim-v7
+++ a/mm/memcontrol.c
@@ -24,8 +24,12 @@
 #include <linux/page-flags.h>
 #include <linux/bit_spinlock.h>
 #include <linux/rcupdate.h>
+#include <linux/swap.h>
+#include <linux/spinlock.h>
+#include <linux/fs.h>

 struct cgroup_subsys mem_cgroup_subsys;
+static const int MEM_CGROUP_RECLAIM_RETRIES = 5;

 /*
  * The memory controller data structure. The memory controller controls both
@@ -51,6 +55,10 @@ struct mem_cgroup {
   */
  struct list_head active_list;
  struct list_head inactive_list;
+ /*
+  * spin_lock to protect the per cgroup LRU
+  */
+ spinlock_t lru_lock;
 };

 /*
@@ -141,6 +149,94 @@ void __always_inline unlock_page_contain
  bit_spin_unlock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
 }

+void __mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
+{
+ if (active)
+  list_move(&pc->lru, &pc->mem_cgroup->active_list);
+ else
+  list_move(&pc->lru, &pc->mem_cgroup->inactive_list);
+}
+
+/*
```

```
+ * This routine assumes that the appropriate zone's lru lock is already held
+ */
+void mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
+{
+ struct mem_cgroup *mem;
+ if (!pc)
+  return;
+
+ mem = pc->mem_cgroup;
+
+ spin_lock(&mem->lru_lock);
+ __mem_cgroup_move_lists(pc, active);
+ spin_unlock(&mem->lru_lock);
+}
+
+unsigned long mem_cgroup_isolate_pages(unsigned long nr_to_scan,
+     struct list_head *dst,
+     unsigned long *scanned, int order,
+     int mode, struct zone *z,
+     struct mem_cgroup *mem_cont,
+     int active)
+{
+ unsigned long nr_taken = 0;
+ struct page *page;
+ unsigned long scan;
+ LIST_HEAD(pc_list);
+ struct list_head *src;
+ struct page_cgroup *pc;
+
+ if (active)
+  src = &mem_cont->active_list;
+ else
+  src = &mem_cont->inactive_list;
+
+ spin_lock(&mem_cont->lru_lock);
+ for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
+  pc = list_entry(src->prev, struct page_cgroup, lru);
+  page = pc->page;
+  VM_BUG_ON(!pc);
+
+  if (PageActive(page) && !active) {
+   __mem_cgroup_move_lists(pc, true);
+   scan--;
+   continue;
+  }
+  if (!PageActive(page) && active) {
+   __mem_cgroup_move_lists(pc, false);
+   scan--;
```

```
+   continue;
+  }
+
+  /*
+   * Reclaim, per zone
+   * TODO: make the active/inactive lists per zone
+   */
+  if (page_zone(page) != z)
+   continue;
+
+  /*
+   * Check if the meta page went away from under us
+   */
+  if (!list_empty(&pc->lru))
+   list_move(&pc->lru, &pc_list);
+  else
+   continue;
+
+  if (__isolate_lru_page(page, mode) == 0) {
+   list_move(&page->lru, dst);
+   nr_taken++;
+  }
+ }
+
+ list_splice(&pc_list, src);
+ spin_unlock(&mem_cont->lru_lock);
+
+ *scanned = scan;
+ return nr_taken;
+}
+
 /*
  * Charge the memory controller for page usage.
  * Return
@@ -151,6 +247,8 @@ int mem_cgroup_charge(struct page *pa
 {
  struct mem_cgroup *mem;
  struct page_cgroup *pc, *race_pc;
+ unsigned long flags;
+ unsigned long nr_retries = MEM_CGROUP_RECLAIM_RETRIES;

  /*
   * Should page_cgroup's go to their own slab?
@@ -197,7 +295,32 @@ int mem_cgroup_charge(struct page *pa
   * If we created the page_cgroup, we should free it on exceeding
   * the cgroup limit.
   */
- if (res_counter_charge(&mem->res, 1)) {
```

```
+ while (res_counter_charge(&mem->res, 1)) {
+  if (try_to_free_mem_cgroup_pages(mem))
+   continue;
+
+  /*
+   * try_to_free_mem_cgroup_pages() might not give us a full
+   * picture of reclaim. Some pages are reclaimed and might be
+   * moved to swap cache or just unmapped from the cgroup.
+   * Check the limit again to see if the reclaim reduced the
+   * current usage of the cgroup before giving up
+   */
+  if (res_counter_check_under_limit(&mem->res))
+   continue;
+   /*
+    * Since we control both RSS and cache, we end up with a
+    * very interesting scenario where we end up reclaiming
+    * memory (essentially RSS), since the memory is pushed
+    * to swap cache, we eventually end up adding those
+    * pages back to our list. Hence we give ourselves a
+    * few chances before we fail
+    */
+  else if (nr_retries--) {
+   congestion_wait(WRITE, HZ/10);
+   continue;
+  }
+
    css_put(&mem->css);
    goto free_pc;
   }
@@ -221,6 +344,10 @@ int mem_cgroup_charge(struct page *pa
   pc->page = page;
   page_assign_page_cgroup(page, pc);

+ spin_lock_irqsave(&mem->lru_lock, flags);
+ list_add(&pc->lru, &mem->active_list);
+ spin_unlock_irqrestore(&mem->lru_lock, flags);
+
  done:
   unlock_page_cgroup(page);
   return 0;
@@ -240,6 +367,7 @@ void mem_cgroup_uncharge(struct page_
  {
   struct mem_cgroup *mem;
   struct page *page;
+ unsigned long flags;

   if (!pc)
    return;
```

```
@@ -252,6 +380,10 @@ void mem_cgroup_uncharge(struct page_
   page_assign_page_cgroup(page, NULL);
   unlock_page_cgroup(page);
   res_counter_uncharge(&mem->res, 1);
+
+   spin_lock_irqsave(&mem->lru_lock, flags);
+   list_del_init(&pc->lru);
+   spin_unlock_irqrestore(&mem->lru_lock, flags);
   kfree(pc);
  }
 }
@@ -310,6 +442,7 @@ mem_cgroup_create(struct cgroup_su
  res_counter_init(&mem->res);
  INIT_LIST_HEAD(&mem->active_list);
  INIT_LIST_HEAD(&mem->inactive_list);
+ spin_lock_init(&mem->lru_lock);
  return &mem->css;
 }

diff -puN mm/swap.c~memory-controller-add-per-cgroup-lru-and-reclaim-v7 mm/swap.c
--- a/mm/swap.c~memory-controller-add-per-cgroup-lru-and-reclaim-v7
+++ a/mm/swap.c
@@ -28,6 +28,7 @@
 #include <linux/percpu.h>
 #include <linux/cpu.h>
 #include <linux/notifier.h>
+#include <linux/memcontrol.h>

 /* How many pages do we try to swap or page in/out together? */
 int page_cluster;
@@ -145,6 +146,7 @@ void fastcall activate_page(struct page
   SetPageActive(page);
   add_page_to_active_list(zone, page);
   __count_vm_event(PGACTIVATE);
+  mem_cgroup_move_lists(page_get_page_cgroup(page), true);
  }
  spin_unlock_irq(&zone->lru_lock);
 }
diff -puN mm/vmscan.c~memory-controller-add-per-cgroup-lru-and-reclaim-v7 mm/vmscan.c
--- a/mm/vmscan.c~memory-controller-add-per-cgroup-lru-and-reclaim-v7
+++ a/mm/vmscan.c
@@ -37,6 +37,7 @@
 #include <linux/delay.h>
 #include <linux/kthread.h>
 #include <linux/freezer.h>
+#include <linux/memcontrol.h>

 #include <asm/tlbflush.h>
```

```
 #include <asm/div64.h>
@@ -68,6 +69,15 @@ struct scan_control {
  int all_unreclaimable;

  int order;
+
+ /* Which cgroup do we reclaim from */
+ struct mem_cgroup *mem_cgroup;
+
+ /* Pluggable isolate pages callback */
+ unsigned long (*isolate_pages)(unsigned long nr, struct list_head *dst,
+   unsigned long *scanned, int order, int mode,
+   struct zone *z, struct mem_cgroup *mem_cont,
+   int active);
 };

 #define lru_to_page(_head) (list_entry((_head)->prev, struct page, lru))
@@ -626,7 +636,7 @@ keep:
  *
  * returns 0 on success, -ve errno on failure.
  */
-static int __isolate_lru_page(struct page *page, int mode)
+int __isolate_lru_page(struct page *page, int mode)
 {
  int ret = -EINVAL;

@@ -760,6 +770,21 @@ static unsigned long isolate_lru_pages(u
  return nr_taken;
 }

+static unsigned long isolate_pages_global(unsigned long nr,
+     struct list_head *dst,
+     unsigned long *scanned, int order,
+     int mode, struct zone *z,
+     struct mem_cgroup *mem_cont,
+     int active)
+{
+ if (active)
+  return isolate_lru_pages(nr, &z->active_list, dst,
+       scanned, order, mode);
+ else
+  return isolate_lru_pages(nr, &z->inactive_list, dst,
+       scanned, order, mode);
+}
+
 /*
  * clear_active_flags() is a helper for shrink_active_list(), clearing
  * any active bits from the pages in the list.
```

```
@@ -801,11 +826,11 @@ static unsigned long shrink_inactive_lis
   unsigned long nr_freed;
   unsigned long nr_active;

-  nr_taken = isolate_lru_pages(sc->swap_cluster_max,
-        &zone->inactive_list,
+  nr_taken = sc->isolate_pages(sc->swap_cluster_max,
         &page_list, &nr_scan, sc->order,
         (sc->order > PAGE_ALLOC_COSTLY_ORDER)?
-          ISOLATE_BOTH : ISOLATE_INACTIVE);
+           ISOLATE_BOTH : ISOLATE_INACTIVE,
+     zone, sc->mem_cgroup, 0);
   nr_active = clear_active_flags(&page_list);
   __count_vm_events(PGDEACTIVATE, nr_active);

@@ -1018,8 +1043,9 @@ force_reclaim_mapped:

  lru_add_drain();
  spin_lock_irq(&zone->lru_lock);
- pgmoved = isolate_lru_pages(nr_pages, &zone->active_list,
-       &l_hold, &pgscanned, sc->order, ISOLATE_ACTIVE);
+ pgmoved = sc->isolate_pages(nr_pages, &l_hold, &pgscanned, sc->order,
+     ISOLATE_ACTIVE, zone,
+     sc->mem_cgroup, 1);
  zone->pages_scanned += pgscanned;
  __mod_zone_page_state(zone, NR_ACTIVE, -pgmoved);
  spin_unlock_irq(&zone->lru_lock);
@@ -1054,6 +1080,7 @@ force_reclaim_mapped:
   ClearPageActive(page);

   list_move(&page->lru, &zone->inactive_list);
+  mem_cgroup_move_lists(page_get_page_cgroup(page), false);
   pgmoved++;
   if (!pagevec_add(&pvec, page)) {
    __mod_zone_page_state(zone, NR_INACTIVE, pgmoved);
@@ -1082,6 +1109,7 @@ force_reclaim_mapped:
   SetPageLRU(page);
   VM_BUG_ON(!PageActive(page));
   list_move(&page->lru, &zone->active_list);
+  mem_cgroup_move_lists(page_get_page_cgroup(page), true);
   pgmoved++;
   if (!pagevec_add(&pvec, page)) {
    __mod_zone_page_state(zone, NR_ACTIVE, pgmoved);
@@ -1213,7 +1241,8 @@ static unsigned long shrink_zones(int pr
  * holds filesystem locks which prevent writeout this might not work, and the
  * allocation attempt will fail.
  */
-unsigned long try_to_free_pages(struct zone **zones, int order, gfp_t gfp_mask)
```

```
+unsigned long do_try_to_free_pages(struct zone **zones, gfp_t gfp_mask,
+     struct scan_control *sc)
 {
  int priority;
  int ret = 0;
@@ -1222,14 +1251,6 @@ unsigned long try_to_free_pages(struct z
  struct reclaim_state *reclaim_state = current->reclaim_state;
  unsigned long lru_pages = 0;
  int i;
- struct scan_control sc = {
-  .gfp_mask = gfp_mask,
-  .may_writepage = !laptop_mode,
-  .swap_cluster_max = SWAP_CLUSTER_MAX,
-  .may_swap = 1,
-  .swappiness = vm_swappiness,
-  .order = order,
- };

  count_vm_event(ALLOCSTALL);

@@ -1244,17 +1265,22 @@ unsigned long try_to_free_pages(struct z
  }

  for (priority = DEF_PRIORITY; priority >= 0; priority--) {
-  sc.nr_scanned = 0;
+  sc->nr_scanned = 0;
   if (!priority)
    disable_swap_token();
-  nr_reclaimed += shrink_zones(priority, zones, &sc);
-  shrink_slab(sc.nr_scanned, gfp_mask, lru_pages);
+  nr_reclaimed += shrink_zones(priority, zones, sc);
+  /*
+   * Don't shrink slabs when reclaiming memory from
+   * over limit cgroups
+   */
+  if (sc->mem_cgroup == NULL)
+   shrink_slab(sc->nr_scanned, gfp_mask, lru_pages);
   if (reclaim_state) {
    nr_reclaimed += reclaim_state->reclaimed_slab;
    reclaim_state->reclaimed_slab = 0;
   }
-  total_scanned += sc.nr_scanned;
-  if (nr_reclaimed >= sc.swap_cluster_max) {
+  total_scanned += sc->nr_scanned;
+  if (nr_reclaimed >= sc->swap_cluster_max) {
    ret = 1;
    goto out;
   }
```

```
@@ -1266,18 +1292,18 @@ unsigned long try_to_free_pages(struct z
    * that's undesirable in laptop mode, where we *want* lumpy
    * writeout.  So in laptop mode, write out the whole world.
    */
-  if (total_scanned > sc.swap_cluster_max +
-      sc.swap_cluster_max / 2) {
+  if (total_scanned > sc->swap_cluster_max +
+      sc->swap_cluster_max / 2) {
     wakeup_pdflush(laptop_mode ? 0 : total_scanned);
-    sc.may_writepage = 1;
+    sc->may_writepage = 1;
   }

   /* Take a nap, wait for some writeback to complete */
-  if (sc.nr_scanned && priority < DEF_PRIORITY - 2)
+  if (sc->nr_scanned && priority < DEF_PRIORITY - 2)
     congestion_wait(WRITE, HZ/10);
  }
  /* top priority shrink_caches still had more to do? don't OOM, then */
- if (!sc.all_unreclaimable)
+ if (!sc->all_unreclaimable && sc->mem_cgroup == NULL)
   ret = 1;
 out:
  /*
@@ -1300,6 +1326,54 @@ out:
  return ret;
 }

+unsigned long try_to_free_pages(struct zone **zones, int order, gfp_t gfp_mask)
+{
+ struct scan_control sc = {
+  .gfp_mask = gfp_mask,
+  .may_writepage = !laptop_mode,
+  .swap_cluster_max = SWAP_CLUSTER_MAX,
+  .may_swap = 1,
+  .swappiness = vm_swappiness,
+  .order = order,
+  .mem_cgroup = NULL,
+  .isolate_pages = isolate_pages_global,
+ };
+
+ return do_try_to_free_pages(zones, gfp_mask, &sc);
+}
+
+#ifdef CONFIG_CGROUP_MEM_CONT
+
+#ifdef CONFIG_HIGHMEM
+#define ZONE_USERPAGES ZONE_HIGHMEM
```

```
+#else
+#define ZONE_USERPAGES ZONE_NORMAL
+#endif
+
+unsigned long try_to_free_mem_cgroup_pages(struct mem_cgroup *mem_cont)
+{
+ struct scan_control sc = {
+  .gfp_mask = GFP_KERNEL,
+  .may_writepage = !laptop_mode,
+  .may_swap = 1,
+  .swap_cluster_max = SWAP_CLUSTER_MAX,
+  .swappiness = vm_swappiness,
+  .order = 1,
+  .mem_cgroup = mem_cont,
+  .isolate_pages = mem_cgroup_isolate_pages,
+ };
+ int node;
+ struct zone **zones;
+
+ for_each_online_node(node) {
+  zones = NODE_DATA(node)->node_zonelists[ZONE_USERPAGES].zones;
+  if (do_try_to_free_pages(zones, sc.gfp_mask, &sc))
+   return 1;
+ }
+ return 0;
+}
+#endif
+
 /*
  * For kswapd, balance_pgdat() will work across all this node's zones until
  * they are all at pages_high.
@@ -1335,6 +1409,8 @@ static unsigned long balance_pgdat(pg_da
   .swap_cluster_max = SWAP_CLUSTER_MAX,
   .swappiness = vm_swappiness,
   .order = order,
+  .mem_cgroup = NULL,
+  .isolate_pages = isolate_pages_global,
  };
  /*
   * temp_priority is used to remember the scanning priority at which
_

--
```

Subject: [PATCH 26/29] memory controller add per container lru and reclaim v7 fix
Posted by Paul Menage on Tue, 11 Sep 2007 19:53:05 GMT
View Forum Message <> Reply to Message

From: Andrew Morton <akpm@linux-foundation.org>

mm/memcontrol.c: In function 'mem_cgroup_charge':
mm/memcontrol.c:337: error: implicit declaration of function 'congestion_wait'

Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Balbir Singh <balbir@linux.vnet.ibm.com>
Cc: David Rientjes <rientjes@google.com>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: Kirill Korotaev <dev@sw.ru>
Cc: Nick Piggin <nickpiggin@yahoo.com.au>
Cc: Paul Menage <menage@google.com>
Cc: Pavel Emelianov <xemul@openvz.org>
Cc: Peter Zijlstra <a.p.zijlstra@chello.nl>
Cc: Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

 mm/memcontrol.c | 1 +
 1 file changed, 1 insertion(+)

diff -puN mm/memcontrol.c~memory-controller-add-per-cgroup-lru-and-reclaim-v7-fix
mm/memcontrol.c
--- a/mm/memcontrol.c~memory-controller-add-per-cgroup-lru-and-reclaim-v7-fix
+++ a/mm/memcontrol.c
@@ -22,6 +22,7 @@
 #include <linux/cgroup.h>
 #include <linux/mm.h>
 #include <linux/page-flags.h>
+#include <linux/backing-dev.h>
 #include <linux/bit_spinlock.h>
 #include <linux/rcupdate.h>
 #include <linux/swap.h>

_

--

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH 27/29] memory controller oom handling v7
Posted by Paul Menage on Tue, 11 Sep 2007 19:53:06 GMT

From: Pavel Emelianov <xemul@openvz.org>

Out of memory handling for cgroups over their limit. A task from the
cgroup over limit is chosen using the existing OOM logic and killed.

TODO:
1. As discussed in the OLS BOF session, consider implementing a user
space policy for OOM handling.

Signed-off-by: Pavel Emelianov <xemul@openvz.org>
Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>
Cc: Paul Menage <menage@google.com>
Cc: Peter Zijlstra <a.p.zijlstra@chello.nl>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Nick Piggin <nickpiggin@yahoo.com.au>
Cc: Kirill Korotaev <dev@sw.ru>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: David Rientjes <rientjes@google.com>
Cc: Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

 include/linux/memcontrol.h | 1
 mm/memcontrol.c | 1
 mm/oom_kill.c | 42 ++++++++++++++++++++++++++++++++++++++----
 3 files changed, 40 insertions(+), 4 deletions(-)

diff -puN include/linux/memcontrol.h~memory-controller-oom-handling-v7
include/linux/memcontrol.h
--- a/include/linux/memcontrol.h~memory-controller-oom-handling-v7
+++ a/include/linux/memcontrol.h
@@ -39,6 +39,7 @@ extern unsigned long mem_cgroup_isola
      int mode, struct zone *z,
      struct mem_cgroup *mem_cont,
      int active);
+extern void mem_cgroup_out_of_memory(struct mem_cgroup *mem);

 static inline void mem_cgroup_uncharge_page(struct page *page)
 {
diff -puN mm/memcontrol.c~memory-controller-oom-handling-v7 mm/memcontrol.c
--- a/mm/memcontrol.c~memory-controller-oom-handling-v7
+++ a/mm/memcontrol.c
@@ -322,6 +322,7 @@ int mem_cgroup_charge(struct page *pa
   }

   css_put(&mem->css);
+ mem_cgroup_out_of_memory(mem);

```
   goto free_pc;
  }

diff -puN mm/oom_kill.c~memory-controller-oom-handling-v7 mm/oom_kill.c
--- a/mm/oom_kill.c~memory-controller-oom-handling-v7
+++ a/mm/oom_kill.c
@@ -25,6 +25,7 @@
 #include <linux/cpuset.h>
 #include <linux/module.h>
 #include <linux/notifier.h>
+#include <linux/memcontrol.h>

 int sysctl_panic_on_oom;
 /* #define DEBUG */
@@ -48,7 +49,8 @@ int sysctl_panic_on_oom;
  *    of least surprise ... (be careful when you change it)
  */

-unsigned long badness(struct task_struct *p, unsigned long uptime)
+unsigned long badness(struct task_struct *p, unsigned long uptime,
+   struct mem_cgroup *mem)
 {
  unsigned long points, cpu_time, run_time, s;
  struct mm_struct *mm;
@@ -61,6 +63,13 @@ unsigned long badness(struct task_struct
   return 0;
  }

+#ifdef CONFIG_CGROUP_MEM_CONT
+ if (mem != NULL && mm->mem_cgroup != mem) {
+  task_unlock(p);
+  return 0;
+ }
+#endif
+
  /*
   * The memory size of the process is the basis for the badness.
   */
@@ -198,7 +207,8 @@ static inline int constrained_alloc(stru
  *
  * (not docbooked, we don't want this one cluttering up the manual)
  */
-static struct task_struct *select_bad_process(unsigned long *ppoints)
+static struct task_struct *select_bad_process(unsigned long *ppoints,
+      struct mem_cgroup *mem)
 {
  struct task_struct *g, *p;
  struct task_struct *chosen = NULL;
```

```
@@ -252,7 +262,7 @@ static struct task_struct *select_bad_pr
   if (p->oomkilladj == OOM_DISABLE)
    continue;

-  points = badness(p, uptime.tv_sec);
+  points = badness(p, uptime.tv_sec, mem);
   if (points > *ppoints || !chosen) {
    chosen = p;
    *ppoints = points;
@@ -364,6 +374,30 @@ static int oom_kill_process(struct task_
  return oom_kill_task(p);
 }

+#ifdef CONFIG_CGROUP_MEM_CONT
+void mem_cgroup_out_of_memory(struct mem_cgroup *mem)
+{
+ unsigned long points = 0;
+ struct task_struct *p;
+
+ cgroup_lock();
+ rcu_read_lock();
+retry:
+ p = select_bad_process(&points, mem);
+ if (PTR_ERR(p) == -1UL)
+  goto out;
+
+ if (!p)
+  p = current;
+
+ if (oom_kill_process(p, points, "Memory cgroup out of memory"))
+  goto retry;
+out:
+ rcu_read_unlock();
+ cgroup_unlock();
+}
+#endif
+
 static BLOCKING_NOTIFIER_HEAD(oom_notify_list);

 int register_oom_notifier(struct notifier_block *nb)
@@ -436,7 +470,7 @@ retry:
    * Rambo mode: Shoot down a process and hope it solves whatever
    * issues we may have.
    */
-  p = select_bad_process(&points);
+  p = select_bad_process(&points, NULL);

   if (PTR_ERR(p) == -1UL)
```

goto out;

–

--

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 28/29] memory controller add switch to control what type of pages to limit v7
Posted by Paul Menage on Tue, 11 Sep 2007 19:53:07 GMT
View Forum Message <> Reply to Message

From: Balbir Singh <balbir@linux.vnet.ibm.com>

Choose if we want cached pages to be accounted or not. By default both are accounted for. A new set of tunables are added.

echo -n 1 > mem_control_type

switches the accounting to account for only mapped pages

echo -n 3 > mem_control_type

switches the behaviour back

Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>
Cc: Pavel Emelianov <xemul@openvz.org>
Cc: Paul Menage <menage@google.com>
Cc: Peter Zijlstra <a.p.zijlstra@chello.nl>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Nick Piggin <nickpiggin@yahoo.com.au>
Cc: Kirill Korotaev <dev@sw.ru>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: David Rientjes <rientjes@google.com>
Cc: Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

include/linux/memcontrol.h | 9 +++
mm/filemap.c | 2
mm/memcontrol.c | 92 ++
mm/swap_state.c | 2
4 files changed, 103 insertions(+), 2 deletions(-)

diff -puN

```
include/linux/memcontrol.h~memory-controller-add-switch-to-control-what-type-of-pages-to-limit-v
7 include/linux/memcontrol.h
---
a/include/linux/memcontrol.h~memory-controller-add-switch-to-control-what-type-of-pages-to-limit-
v7
+++ a/include/linux/memcontrol.h
@@ -20,6 +20,8 @@
 #ifndef _LINUX_MEMCONTROL_H
 #define _LINUX_MEMCONTROL_H

+#include <linux/mm.h>
+
 struct mem_cgroup;
 struct page_cgroup;

@@ -40,6 +42,7 @@ extern unsigned long mem_cgroup_isola
       struct mem_cgroup *mem_cont,
       int active);
 extern void mem_cgroup_out_of_memory(struct mem_cgroup *mem);
+extern int mem_cgroup_cache_charge(struct page *page, struct mm_struct *mm);

 static inline void mem_cgroup_uncharge_page(struct page *page)
 {
@@ -84,6 +87,12 @@ static inline void mem_cgroup_move_li
 {
 }

+static inline int mem_cgroup_cache_charge(struct page *page,
+       struct mm_struct *mm)
+{
+ return 0;
+}
+
 #endif /* CONFIG_CGROUP_MEM_CONT */

 #endif /* _LINUX_MEMCONTROL_H */
diff -puN mm/filemap.c~memory-controller-add-switch-to-control-what-type-of-pages-to-limit-v7
mm/filemap.c
--- a/mm/filemap.c~memory-controller-add-switch-to-control-what-type-of-pages-to-limit-v7
+++ a/mm/filemap.c
@@ -444,7 +444,7 @@ int add_to_page_cache(struct page *page,

   if (error == 0) {

-  error = mem_cgroup_charge(page, current->mm);
+   error = mem_cgroup_cache_charge(page, current->mm);
    if (error)
     goto out;
```

```
diff -puN
mm/memcontrol.c~memory-controller-add-switch-to-control-what-type-of-pages-to-limit-v7
mm/memcontrol.c
--- a/mm/memcontrol.c~memory-controller-add-switch-to-control-what-type-of-pages-to-limit-v7
+++ a/mm/memcontrol.c
@@ -28,6 +28,8 @@
 #include <linux/spinlock.h>
 #include <linux/fs.h>

+#include <asm/uaccess.h>
+
 struct cgroup_subsys mem_cgroup_subsys;
 static const int MEM_CGROUP_RECLAIM_RETRIES = 5;

@@ -59,6 +61,7 @@ struct mem_cgroup {
   * spin_lock to protect the per cgroup LRU
   */
  spinlock_t lru_lock;
+ unsigned long control_type; /* control RSS or RSS+Pagecache */
 };

 /*
@@ -81,6 +84,15 @@ struct page_cgroup {
      /* mapped and cached states     */
 };

+enum {
+ MEM_CGROUP_TYPE_UNSPEC = 0,
+ MEM_CGROUP_TYPE_MAPPED,
+ MEM_CGROUP_TYPE_CACHED,
+ MEM_CGROUP_TYPE_ALL,
+ MEM_CGROUP_TYPE_MAX,
+} mem_control_type;
+
+static struct mem_cgroup init_mem_cgroup;

 static inline
 struct mem_cgroup *mem_cgroup_from_cont(struct cgroup *cont)
@@ -361,6 +373,22 @@ err:
 }

 /*
+ * See if the cached pages should be charged at all?
+ */
+int mem_cgroup_cache_charge(struct page *page, struct mm_struct *mm)
+{
+ struct mem_cgroup *mem;
```

```
+ if (!mm)
+  mm = &init_mm;
+
+ mem = rcu_dereference(mm->mem_cgroup);
+ if (mem->control_type == MEM_CGROUP_TYPE_ALL)
+  return mem_cgroup_charge(page, mm);
+ else
+  return 0;
+}
+
+/*
  * Uncharging is always a welcome operation, we never complain, simply
  * uncharge.
  */
@@ -370,6 +398,10 @@ void mem_cgroup_uncharge(struct page_
  struct page *page;
  unsigned long flags;

+ /*
+  * This can handle cases when a page is not charged at all and we
+  * are switching between handling the control_type.
+  */
  if (!pc)
   return;

@@ -405,6 +437,60 @@ static ssize_t mem_cgroup_write(struc
     cft->private, userbuf, nbytes, ppos);
 }

+static ssize_t mem_control_type_write(struct cgroup *cont,
+   struct cftype *cft, struct file *file,
+   const char __user *userbuf,
+   size_t nbytes, loff_t *pos)
+{
+ int ret;
+ char *buf, *end;
+ unsigned long tmp;
+ struct mem_cgroup *mem;
+
+ mem = mem_cgroup_from_cont(cont);
+ buf = kmalloc(nbytes + 1, GFP_KERNEL);
+ ret = -ENOMEM;
+ if (buf == NULL)
+  goto out;
+
+ buf[nbytes] = 0;
+ ret = -EFAULT;
+ if (copy_from_user(buf, userbuf, nbytes))
```

```
+  goto out_free;
+
+ ret = -EINVAL;
+ tmp = simple_strtoul(buf, &end, 10);
+ if (*end != '\0')
+  goto out_free;
+
+ if (tmp <= MEM_CGROUP_TYPE_UNSPEC || tmp >= MEM_CGROUP_TYPE_MAX)
+  goto out_free;
+
+ mem->control_type = tmp;
+ ret = nbytes;
+out_free:
+ kfree(buf);
+out:
+ return ret;
+}
+
+static ssize_t mem_control_type_read(struct cgroup *cont,
+    struct cftype *cft,
+    struct file *file, char __user *userbuf,
+    size_t nbytes, loff_t *ppos)
+{
+ unsigned long val;
+ char buf[64], *s;
+ struct mem_cgroup *mem;
+
+ mem = mem_cgroup_from_cont(cont);
+ s = buf;
+ val = mem->control_type;
+ s += sprintf(s, "%lu\n", val);
+ return simple_read_from_buffer((void __user *)userbuf, nbytes,
+   ppos, buf, s - buf);
+}
+
 static struct cftype mem_cgroup_files[] = {
  {
   .name = "usage",
@@ -422,6 +508,11 @@ static struct cftype mem_cgroup_files
   .private = RES_FAILCNT,
   .read = mem_cgroup_read,
  },
+ {
+  .name = "control_type",
+  .write = mem_control_type_write,
+  .read = mem_control_type_read,
+ },
 };
```

```
 static struct mem_cgroup init_mem_cgroup;
@@ -444,6 +535,7 @@ mem_cgroup_create(struct cgroup_su
  INIT_LIST_HEAD(&mem->active_list);
  INIT_LIST_HEAD(&mem->inactive_list);
  spin_lock_init(&mem->lru_lock);
+ mem->control_type = MEM_CGROUP_TYPE_ALL;
  return &mem->css;
 }

diff -puN
mm/swap_state.c~memory-controller-add-switch-to-control-what-type-of-pages-to-limit-v7
mm/swap_state.c
--- a/mm/swap_state.c~memory-controller-add-switch-to-control-what-type-of-pages-to-limit-v7
+++ a/mm/swap_state.c
@@ -81,7 +81,7 @@ static int __add_to_swap_cache(struct pa
  error = radix_tree_preload(gfp_mask);
  if (!error) {

-  error = mem_cgroup_charge(page, current->mm);
+  error = mem_cgroup_cache_charge(page, current->mm);
   if (error)
    goto out;


_

--
_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers
```

Subject: [PATCH 29/29] memory controller make page_referenced container aware v7

Posted by Paul Menage on Tue, 11 Sep 2007 19:53:08 GMT

View Forum Message <> Reply to Message

From: Balbir Singh <balbir@linux.vnet.ibm.com>

Make page_referenced() cgroup aware. Without this patch, page_referenced()
can cause a page to be skipped while reclaiming pages. This patch ensures
that other cgroups do not hold pages in a particular cgroup hostage. It
is required to ensure that shared pages are freed from a cgroup when they
are not actively referenced from the cgroup that brought them in

Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>
Cc: Pavel Emelianov <xemul@openvz.org>

```
Cc: Paul Menage <menage@google.com>
Cc: Peter Zijlstra <a.p.zijlstra@chello.nl>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: Nick Piggin <nickpiggin@yahoo.com.au>
Cc: Kirill Korotaev <dev@sw.ru>
Cc: Herbert Poetzl <herbert@13thfloor.at>
Cc: David Rientjes <rientjes@google.com>
Cc: Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

 include/linux/memcontrol.h |    6 ++++++
 include/linux/rmap.h       |    5 +++--
 mm/memcontrol.c            |    5 +++++
 mm/rmap.c                  |   30 ++++++++++++++++++++++++++------
 mm/vmscan.c                |    4 ++--
 5 files changed, 40 insertions(+), 10 deletions(-)

diff -puN include/linux/memcontrol.h~memory-controller-make-page_referenced-cgroup-aware-v7
include/linux/memcontrol.h
--- a/include/linux/memcontrol.h~memory-controller-make-page_referenced-cgroup-aware-v7
+++ a/include/linux/memcontrol.h
@@ -43,6 +43,7 @@ extern unsigned long mem_cgroup_isola
      int active);
 extern void mem_cgroup_out_of_memory(struct mem_cgroup *mem);
 extern int mem_cgroup_cache_charge(struct page *page, struct mm_struct *mm);
+extern struct mem_cgroup *mm_cgroup(struct mm_struct *mm);

 static inline void mem_cgroup_uncharge_page(struct page *page)
 {
@@ -93,6 +94,11 @@ static inline int mem_cgroup_cache_ch
  return 0;
 }

+static inline struct mem_cgroup *mm_cgroup(struct mm_struct *mm)
+{
+ return NULL;
+}
+
 #endif /* CONFIG_CGROUP_MEM_CONT */

 #endif /* _LINUX_MEMCONTROL_H */
diff -puN include/linux/rmap.h~memory-controller-make-page_referenced-cgroup-aware-v7
include/linux/rmap.h
--- a/include/linux/rmap.h~memory-controller-make-page_referenced-cgroup-aware-v7
+++ a/include/linux/rmap.h
@@ -8,6 +8,7 @@
 #include <linux/slab.h>
```

```
 #include <linux/mm.h>
 #include <linux/spinlock.h>
+#include <linux/memcontrol.h>

 /*
  * The anon_vma heads a list of private "related" vmas, to scan if
@@ -86,7 +87,7 @@ static inline void page_dup_rmap(struct
 /*
  * Called from mm/vmscan.c to handle paging out
  */
-int page_referenced(struct page *, int is_locked);
+int page_referenced(struct page *, int is_locked, struct mem_cgroup *cnt);
 int try_to_unmap(struct page *, int ignore_refs);

 /*
@@ -114,7 +115,7 @@ int page_mkclean(struct page *);
 #define anon_vma_prepare(vma) (0)
 #define anon_vma_link(vma) do {} while (0)

-#define page_referenced(page,l) TestClearPageReferenced(page)
+#define page_referenced(page,l,cnt) TestClearPageReferenced(page)
 #define try_to_unmap(page, refs) SWAP_FAIL

 static inline int page_mkclean(struct page *page)
diff -puN mm/memcontrol.c~memory-controller-make-page_referenced-cgroup-aware-v7
mm/memcontrol.c
--- a/mm/memcontrol.c~memory-controller-make-page_referenced-cgroup-aware-v7
+++ a/mm/memcontrol.c
@@ -109,6 +109,11 @@ struct mem_cgroup *mem_cgroup_from
     struct mem_cgroup, css);
 }

+inline struct mem_cgroup *mm_cgroup(struct mm_struct *mm)
+{
+ return rcu_dereference(mm->mem_cgroup);
+}
+
 void mm_init_cgroup(struct mm_struct *mm, struct task_struct *p)
 {
  struct mem_cgroup *mem;
diff -puN mm/rmap.c~memory-controller-make-page_referenced-cgroup-aware-v7 mm/rmap.c
--- a/mm/rmap.c~memory-controller-make-page_referenced-cgroup-aware-v7
+++ a/mm/rmap.c
@@ -299,7 +299,8 @@ out:
  return referenced;
 }

-static int page_referenced_anon(struct page *page)
```

```
+static int page_referenced_anon(struct page *page,
+    struct mem_cgroup *mem_cont)
 {
  unsigned int mapcount;
  struct anon_vma *anon_vma;
@@ -312,6 +313,13 @@ static int page_referenced_anon(struct p

  mapcount = page_mapcount(page);
  list_for_each_entry(vma, &anon_vma->head, anon_vma_node) {
+  /*
+   * If we are reclaiming on behalf of a cgroup, skip
+   * counting on behalf of references from different
+   * cgroups
+   */
+  if (mem_cont && (mm_cgroup(vma->vm_mm) != mem_cont))
+   continue;
    referenced += page_referenced_one(page, vma, &mapcount);
    if (!mapcount)
     break;
@@ -332,7 +340,8 @@ static int page_referenced_anon(struct p
  *
  * This function is only called from page_referenced for object-based pages.
  */
-static int page_referenced_file(struct page *page)
+static int page_referenced_file(struct page *page,
+    struct mem_cgroup *mem_cont)
 {
  unsigned int mapcount;
  struct address_space *mapping = page->mapping;
@@ -365,6 +374,13 @@ static int page_referenced_file(struct p
  mapcount = page_mapcount(page);

  vma_prio_tree_foreach(vma, &iter, &mapping->i_mmap, pgoff, pgoff) {
+  /*
+   * If we are reclaiming on behalf of a cgroup, skip
+   * counting on behalf of references from different
+   * cgroups
+   */
+  if (mem_cont && (mm_cgroup(vma->vm_mm) != mem_cont))
+   continue;
    if ((vma->vm_flags & (VM_LOCKED|VM_MAYSHARE))
        == (VM_LOCKED|VM_MAYSHARE)) {
     referenced++;
@@ -387,7 +403,8 @@ static int page_referenced_file(struct p
  * Quick test_and_clear_referenced for all mappings to a page,
  * returns the number of ptes which referenced the page.
  */
-int page_referenced(struct page *page, int is_locked)
```

```
+int page_referenced(struct page *page, int is_locked,
+   struct mem_cgroup *mem_cont)
 {
  int referenced = 0;

@@ -399,14 +416,15 @@ int page_referenced(struct page *page, i

  if (page_mapped(page) && page->mapping) {
   if (PageAnon(page))
-   referenced += page_referenced_anon(page);
+   referenced += page_referenced_anon(page, mem_cont);
   else if (is_locked)
-   referenced += page_referenced_file(page);
+   referenced += page_referenced_file(page, mem_cont);
   else if (TestSetPageLocked(page))
    referenced++;
   else {
    if (page->mapping)
-    referenced += page_referenced_file(page);
+    referenced +=
+     page_referenced_file(page, mem_cont);
    unlock_page(page);
   }
  }
diff -puN mm/vmscan.c~memory-controller-make-page_referenced-cgroup-aware-v7
mm/vmscan.c
--- a/mm/vmscan.c~memory-controller-make-page_referenced-cgroup-aware-v7
+++ a/mm/vmscan.c
@@ -503,7 +503,7 @@ static unsigned long shrink_page_list(st
     goto keep_locked;
   }

-  referenced = page_referenced(page, 1);
+  referenced = page_referenced(page, 1, sc->mem_cgroup);
   /* In active use or really unfreeable?  Activate it. */
   if (sc->order <= PAGE_ALLOC_COSTLY_ORDER &&
     referenced && page_mapping_inuse(page))
@@ -1057,7 +1057,7 @@ force_reclaim_mapped:
   if (page_mapped(page)) {
    if (!reclaim_mapped ||
        (total_swap_pages == 0 && PageAnon(page)) ||
-       page_referenced(page, 0)) {
+       page_referenced(page, 0, sc->mem_cgroup)) {
     list_add(&page->lru, &l_active);
     continue;
    }
_
```

--

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 23/29] memory controller memory accounting v7
Posted by Peter Zijlstra on Wed, 12 Sep 2007 20:56:50 GMT
View Forum Message <> Reply to Message

> From: Balbir Singh <balbir@linux.vnet.ibm.com>

> void page_assign_page_cgroup(struct page *page, struct page_cgroup *pc)
> {
> - page->page_cgroup = (unsigned long)pc;
> + int locked;
> +
> + /*
> + * While resetting the page_cgroup we might not hold the
> + * page_cgroup lock. free_hot_cold_page() is an example
> + * of such a scenario
> + */
> + if (pc)
> + VM_BUG_ON(!page_cgroup_locked(page));
> + locked = (page->page_cgroup & PAGE_CGROUP_LOCK);
> + page->page_cgroup = ((unsigned long)pc | locked);
> }

This looks a bit odd, why not write:

 locked = page_cgroup_locked(page);
 if (pc)
   VM_BUG_ON(!locked)

> +/*
> + * Charge the memory controller for page usage.
> + * Return
> + * 0 if the charge was successful
> + * < 0 if the cgroup is over its limit
> + */
> +int mem_cgroup_charge(struct page *page, struct mm_struct *mm)
> +{
> + struct mem_cgroup *mem;
> + struct page_cgroup *pc, *race_pc;
> +
> + /*
> + * Should page_cgroup's go to their own slab?

```
> +  * One could optimize the performance of the charging routine
> +  * by saving a bit in the page_flags and using it as a lock
> +  * to see if the cgroup page already has a page_cgroup associated
> +  * with it
> +  */
> + lock_page_cgroup(page);
> + pc = page_get_page_cgroup(page);
> + /*
> +  * The page_cgroup exists and the page has already been accounted
> +  */
> + if (pc) {
> +  atomic_inc(&pc->ref_cnt);
> +  goto done;
> + }
> +
> + unlock_page_cgroup(page);
> +
> + pc = kzalloc(sizeof(struct page_cgroup), GFP_KERNEL);
> + if (pc == NULL)
> +  goto err;
> +
> + rcu_read_lock();
> + /*
> +  * We always charge the cgroup the mm_struct belongs to
> +  * the mm_struct's mem_cgroup changes on task migration if the
> +  * thread group leader migrates. It's possible that mm is not
> +  * set, if so charge the init_mm (happens for pagecache usage).
> +  */
> + if (!mm)
> +  mm = &init_mm;
> +
> + mem = rcu_dereference(mm->mem_cgroup);
> + /*
> +  * For every charge from the cgroup, increment reference
> +  * count
> +  */
> + css_get(&mem->css);
> + rcu_read_unlock();
> +
> + /*
> +  * If we created the page_cgroup, we should free it on exceeding
> +  * the cgroup limit.
> +  */
> + if (res_counter_charge(&mem->res, 1)) {
> +  css_put(&mem->css);
> +  goto free_pc;
> + }
> +
```

> + lock_page_cgroup(page);
> + /*
> + * Check if somebody else beat us to allocating the page_cgroup
> + */
> + race_pc = page_get_page_cgroup(page);
> + if (race_pc) {
> + kfree(pc);
> + pc = race_pc;
> + atomic_inc(&pc->ref_cnt);

This inc

> + res_counter_uncharge(&mem->res, 1);
> + css_put(&mem->css);
> + goto done;
> + }
> +
> + atomic_set(&pc->ref_cnt, 1);

combined with this set make me wonder...

> + pc->mem_cgroup = mem;
> + pc->page = page;
> + page_assign_page_cgroup(page, pc);
> +
> +done:
> + unlock_page_cgroup(page);
> + return 0;
> +free_pc:
> + kfree(pc);
> + return -ENOMEM;
> +err:
> + unlock_page_cgroup(page);
> + return -ENOMEM;
> +}

> @@ -2161,6 +2184,9 @@ static int do_anonymous_page(struct mm_s
>   if (!page)
>   goto oom;
>
> + if (mem_cgroup_charge(page, mm))
> +   goto oom_free_page;
> +
>   entry = mk_pte(page, vma->vm_page_prot);
>   entry = maybe_mkwrite(pte_mkdirty(entry), vma);
>

whitespace damage

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 23/29] memory controller memory accounting v7
Posted by Balbir Singh on Thu, 13 Sep 2007 09:49:01 GMT
View Forum Message <> Reply to Message

Peter Zijlstra wrote:
>> From: Balbir Singh <balbir@linux.vnet.ibm.com>
>
>> void page_assign_page_cgroup(struct page *page, struct page_cgroup *pc)
>> {
>> - page->page_cgroup = (unsigned long)pc;
>> + int locked;
>> +
>> + /*
>> + * While resetting the page_cgroup we might not hold the
>> + * page_cgroup lock. free_hot_cold_page() is an example
>> + * of such a scenario
>> + */
>> + if (pc)
>> + VM_BUG_ON(!page_cgroup_locked(page));
>> + locked = (page->page_cgroup & PAGE_CGROUP_LOCK);
>> + page->page_cgroup = ((unsigned long)pc | locked);
>> }
>
> This looks a bit odd, why not write:
>
> locked = page_cgroup_locked(page);
> if (pc)
>   VM_BUG_ON(!locked)
>

Sure, we could write it this way or

VM_BUG_ON(pc && !locked)

>> +/*
>> + * Charge the memory controller for page usage.

```
>> + * Return
>> + * 0 if the charge was successful
>> + * < 0 if the cgroup is over its limit
>> + */
>> +int mem_cgroup_charge(struct page *page, struct mm_struct *mm)
>> +{
>> + struct mem_cgroup *mem;
>> + struct page_cgroup *pc, *race_pc;
>> +
>> + /*
>> +  * Should page_cgroup's go to their own slab?
>> +  * One could optimize the performance of the charging routine
>> +  * by saving a bit in the page_flags and using it as a lock
>> +  * to see if the cgroup page already has a page_cgroup associated
>> +  * with it
>> +  */
>> + lock_page_cgroup(page);
>> + pc = page_get_page_cgroup(page);
>> + /*
>> +  * The page_cgroup exists and the page has already been accounted
>> +  */
>> + if (pc) {
>> +  atomic_inc(&pc->ref_cnt);
>> +  goto done;
>> + }
>> +
>> + unlock_page_cgroup(page);
>> +
>> + pc = kzalloc(sizeof(struct page_cgroup), GFP_KERNEL);
>> + if (pc == NULL)
>> +  goto err;
>> +
>> + rcu_read_lock();
>> + /*
>> +  * We always charge the cgroup the mm_struct belongs to
>> +  * the mm_struct's mem_cgroup changes on task migration if the
>> +  * thread group leader migrates. It's possible that mm is not
>> +  * set, if so charge the init_mm (happens for pagecache usage).
>> +  */
>> + if (!mm)
>> +  mm = &init_mm;
>> +
>> + mem = rcu_dereference(mm->mem_cgroup);
>> + /*
>> +  * For every charge from the cgroup, increment reference
>> +  * count
>> +  */
>> + css_get(&mem->css);
```

>> + rcu_read_unlock();
>> +
>> + /*
>> + * If we created the page_cgroup, we should free it on exceeding
>> + * the cgroup limit.
>> + */
>> + if (res_counter_charge(&mem->res, 1)) {
>> + css_put(&mem->css);
>> + goto free_pc;
>> + }
>> +
>> + lock_page_cgroup(page);
>> + /*
>> + * Check if somebody else beat us to allocating the page_cgroup
>> + */
>> + race_pc = page_get_page_cgroup(page);
>> + if (race_pc) {
>> + kfree(pc);
>> + pc = race_pc;
>> + atomic_inc(&pc->ref_cnt);
>
> This inc
>
>> + res_counter_uncharge(&mem->res, 1);
>> + css_put(&mem->css);
>> + goto done;
>> + }
>> +
>> + atomic_set(&pc->ref_cnt, 1);
>
> combined with this set make me wonder...
>

I am not sure I understand this comment.

>> + pc->mem_cgroup = mem;
>> + pc->page = page;
>> + page_assign_page_cgroup(page, pc);
>> +
>> +done:
>> + unlock_page_cgroup(page);
>> + return 0;
>> +free_pc:
>> + kfree(pc);
>> + return -ENOMEM;
>> +err:
>> + unlock_page_cgroup(page);
>> + return -ENOMEM;

>> +}
>
>
>
>> @@ -2161,6 +2184,9 @@ static int do_anonymous_page(struct mm_s
>>   if (!page)
>>   goto oom;
>>
>> + if (mem_cgroup_charge(page, mm))
>> +   goto oom_free_page;
>> +
>>   entry = mk_pte(page, vma->vm_page_prot);
>>   entry = maybe_mkwrite(pte_mkdirty(entry), vma);
>>
>
> whitespace damage
>

Yes, it's already been fixed in Andrew's tree. Paul could you
please pull in the those fixes as well? They are not in a -mm
tree, but you can find them on mm-commits.

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 23/29] memory controller memory accounting v7
Posted by Peter Zijlstra on Thu, 13 Sep 2007 10:18:01 GMT
View Forum Message <> Reply to Message

On Thu, 2007-09-13 at 15:19 +0530, Balbir Singh wrote:

> VM_BUG_ON(pc && !locked)

Even better :-)

> >> +/*
> >> + * Charge the memory controller for page usage.
> >> + * Return

```
> >> + * 0 if the charge was successful
> >> + * < 0 if the cgroup is over its limit
> >> + */
> >> +int mem_cgroup_charge(struct page *page, struct mm_struct *mm)
> >> +{
> >> + struct mem_cgroup *mem;
> >> + struct page_cgroup *pc, *race_pc;
> >> +
> >> + /*
> >> +  * Should page_cgroup's go to their own slab?
> >> +  * One could optimize the performance of the charging routine
> >> +  * by saving a bit in the page_flags and using it as a lock
> >> +  * to see if the cgroup page already has a page_cgroup associated
> >> +  * with it
> >> +  */
> >> + lock_page_cgroup(page);
> >> + pc = page_get_page_cgroup(page);
> >> + /*
> >> +  * The page_cgroup exists and the page has already been accounted
> >> +  */
> >> + if (pc) {
> >> +  atomic_inc(&pc->ref_cnt);
> >> +  goto done;
> >> + }
> >> +
> >> + unlock_page_cgroup(page);
> >> +
> >> + pc = kzalloc(sizeof(struct page_cgroup), GFP_KERNEL);
> >> + if (pc == NULL)
> >> +  goto err;
> >> +
> >> + rcu_read_lock();
> >> + /*
> >> +  * We always charge the cgroup the mm_struct belongs to
> >> +  * the mm_struct's mem_cgroup changes on task migration if the
> >> +  * thread group leader migrates. It's possible that mm is not
> >> +  * set, if so charge the init_mm (happens for pagecache usage).
> >> +  */
> >> + if (!mm)
> >> +  mm = &init_mm;
> >> +
> >> + mem = rcu_dereference(mm->mem_cgroup);
> >> + /*
> >> +  * For every charge from the cgroup, increment reference
> >> +  * count
> >> +  */
> >> + css_get(&mem->css);
> >> + rcu_read_unlock();
```

> >> +
> >> + /*
> >> + * If we created the page_cgroup, we should free it on exceeding
> >> + * the cgroup limit.
> >> + */
> >> + if (res_counter_charge(&mem->res, 1)) {
> >> + css_put(&mem->css);
> >> + goto free_pc;
> >> + }
> >> +
> >> + lock_page_cgroup(page);
> >> + /*
> >> + * Check if somebody else beat us to allocating the page_cgroup
> >> + */
> >> + race_pc = page_get_page_cgroup(page);
> >> + if (race_pc) {
> >> + kfree(pc);
> >> + pc = race_pc;
> >> + atomic_inc(&pc->ref_cnt);
> >
> > This inc
> >
> >> + res_counter_uncharge(&mem->res, 1);
> >> + css_put(&mem->css);
> >> + goto done;
> >> + }
> >> +
> >> + atomic_set(&pc->ref_cnt, 1);
> >
> > combined with this set make me wonder...
> >
>
> I am not sure I understand this comment.

Is that inc needed? the pc is already associated with the page and
should thus already have a reference, so this inc would do 1->2, but we
then set it to 1 again. seems like a superfluous operation.

Subject: Re: [PATCH 23/29] memory controller memory accounting v7
Posted by Balbir Singh on Thu, 13 Sep 2007 10:29:08 GMT
View Forum Message <> Reply to Message

Peter Zijlstra wrote:
> On Thu, 2007-09-13 at 15:19 +0530, Balbir Singh wrote:
>
>> VM_BUG_ON(pc && !locked)
>
> Even better :-)
>

Good, I'll change it.

>> I am not sure I understand this comment.
>
> Is that inc needed? the pc is already associated with the page and
> should thus already have a reference, so this inc would do 1->2, but we
> then set it to 1 again. seems like a superfluous operation.

That is something I need to look into. I'll try and get to it soon.

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 01/29] task containersv11 basic task container framework
Posted by Paul Jackson on Sun, 30 Sep 2007 04:40:43 GMT
View Forum Message <> Reply to Message

Paul M:

This patch doesn't build for me in the following case. If I apply the
rest of the containersv11 patches, it builds, but if I happen to bisect
into this set of patches having applied only:

task-containersv11-basic-task-container-framework.patch

while using sn2_defconfig (with CONFIG_CGROUPS=y), then a build for
arch ia64 fails with:

$ make kernel/cgroup.o
CHK include/linux/version.h

```
  CHK     include/linux/utsrelease.h
  CALL    scripts/checksyscalls.sh
<stdin>:1389:2: warning: #warning syscall revokeat not implemented
<stdin>:1393:2: warning: #warning syscall frevoke not implemented
  CC      kernel/cgroup.o
kernel/cgroup.c: In function 'cgroup_new_inode':
kernel/cgroup.c:227: error: variable 'cgroup_backing_dev_info' has initializer but incomplete type
kernel/cgroup.c:228: error: unknown field 'capabilities' specified in initializer
kernel/cgroup.c:228: error: 'BDI_CAP_NO_ACCT_DIRTY' undeclared (first use in this function)
kernel/cgroup.c:228: error: (Each undeclared identifier is reported only once
kernel/cgroup.c:228: error: for each function it appears in.)
kernel/cgroup.c:228: error: 'BDI_CAP_NO_WRITEBACK' undeclared (first use in this function)
kernel/cgroup.c:228: warning: excess elements in struct initializer
kernel/cgroup.c:228: warning: (near initialization for 'cgroup_backing_dev_info')
kernel/cgroup.c:227: error: storage size of 'cgroup_backing_dev_info' isn't known
kernel/cgroup.c:227: warning: unused variable 'cgroup_backing_dev_info'
make[1]: *** [kernel/cgroup.o] Error 1
make: *** [kernel/cgroup.o] Error 2
```

I haven't tested building other partial applications of the
containersv11 patch set; any initial sequence of them should build.

... also ... too bad the names of these patches all have 'container' in
them, not 'cgroup'. I realize how this came to be, due to changing
container to cgroup after these patches were named, but it sure would
be nice if the final patch set record that gets layed down in Linus's
history showed the correct name of 'cgroup' in these patch names.

Well ... nice in my view ... such a patch rename would create more work
for Andrew, so I can't claim to know what he will think of this patch
name detail.

--

I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.925.600.0401

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 01/29] task containersv11 basic task container framework
Posted by Paul Jackson on Sun, 30 Sep 2007 05:10:30 GMT
View Forum Message <> Reply to Message

I had to push down all of the following patches, before I could get it to build:

task-containersv11-basic-task-container-framework.patch
task-containersv11-basic-task-container-framework-fix.patch
task-containersv11-basic-task-container-framework-containers-fix-refcount-bug.patch
task-containersv11-add-tasks-file-interface.patch
task-containersv11-add-fork-exit-hooks.patch
task-containersv11-add-container_clone-interface.patch
task-containersv11-add-container_clone-interface-containers-fix-refcount-bug.patch
task-containersv11-add-procfs-interface.patch
task-containersv11-add-procfs-interface-containers-bdi-init-hooks.patch
task-containersv11-shared-container-subsystem-group-arrays.patch
task-containersv11-shared-container-subsystem-group-arrays-avoid-lockdep-warning.patch
task-containersv11-shared-container-subsystem-group-arrays-include-fix.patch

That's a big stretch of forbidden territory for innocent bisecters.

--

I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.925.600.0401

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 01/29] task containersv11 basic task container framework
Posted by Paul Jackson on Sun, 30 Sep 2007 05:14:25 GMT
View Forum Message <> Reply to Message

And then I pushed just one more patch:

task-containersv11-automatic-userspace-notification-of-idle-containers.patch

and the build died again:

```
$ make kernel/cgroup.o
  CHK     include/linux/version.h
  CHK     include/linux/utsrelease.h
  CALL    scripts/checksyscalls.sh
<stdin>:1389:2: warning: #warning syscall revokeat not implemented
<stdin>:1393:2: warning: #warning syscall frevoke not implemented
  CC      kernel/cgroup.o
kernel/cgroup.c: In function 'cgroup_release_agent':
kernel/cgroup.c:2725: error: implicit declaration of function 'call_usermodehelper'
kernel/cgroup.c:2725: error: 'UMH_WAIT_EXEC' undeclared (first use in this function)
kernel/cgroup.c:2725: error: (Each undeclared identifier is reported only once
kernel/cgroup.c:2725: error: for each function it appears in.)
```

make[1]: *** [kernel/cgroup.o] Error 1
make: *** [kernel/cgroup.o] Error 2

--
I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.925.600.0401

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 11/29] task containersv11 make cpusets a client of containers
Posted by Paul Jackson on Sun, 30 Sep 2007 06:25:13 GMT
View Forum Message <> Reply to Message

There are some 21 mentions of 'manage_mutex' in the comments of
kernel/cpuset.c remaining after this patch is applied, but no such
mutex exists anymore.

Could you update kernel/cpuset.c comments, Paul M., for this and
whatever other changes apply ?

--
I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.925.600.0401

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 01/29] task containersv11 basic task container framework
Posted by Paul Menage on Sun, 30 Sep 2007 07:10:18 GMT
View Forum Message <> Reply to Message

On 9/29/07, Paul Jackson <pj@sgi.com> wrote:
> Paul M:
>
> This patch doesn't build for me in the following case. If I apply the
> rest of the containersv11 patches, it builds, but if I happen to bisect
> into this set of patches having applied only:

These patches weren't the normal style of incremental patchset - they

were direct substitutions of the previous patches in Andrew's -mm
tree.

Andrew's approach when encountering patches that are, e.g. missing
includes on some architectures, is to add a *-fix.patch to the series;
in this case it looks as though the fix for the missing ia64 include
was added towards the end of the series rather than directly after the
patch that needed it. Hence the fact that you couldn't compile until
that later patch had been pushed.

Andrew, is it practical for you to collapse any *-fix.patch files for
cgroups into the appropriate patches that they fix? That would
simplify the patch set a bit.

>
>   task-containersv11-basic-task-container-framework.patch
>
> while using sn2_defconfig (with CONFIG_CGROUPS=y), then a build for
> arch ia64 fails with:
>
>
> $ make kernel/cgroup.o
>   CHK   include/linux/version.h
>   CHK   include/linux/utsrelease.h
>   CALL   scripts/checksyscalls.sh
> <stdin>:1389:2: warning: #warning syscall revokeat not implemented
> <stdin>:1393:2: warning: #warning syscall frevoke not implemented
>   CC   kernel/cgroup.o
> kernel/cgroup.c: In function 'cgroup_new_inode':
> kernel/cgroup.c:227: error: variable 'cgroup_backing_dev_info' has initializer but incomplete type
> kernel/cgroup.c:228: error: unknown field 'capabilities' specified in initializer
> kernel/cgroup.c:228: error: 'BDI_CAP_NO_ACCT_DIRTY' undeclared (first use in this function)
> kernel/cgroup.c:228: error: (Each undeclared identifier is reported only once
> kernel/cgroup.c:228: error: for each function it appears in.)
> kernel/cgroup.c:228: error: 'BDI_CAP_NO_WRITEBACK' undeclared (first use in this function)
> kernel/cgroup.c:228: warning: excess elements in struct initializer
> kernel/cgroup.c:228: warning: (near initialization for 'cgroup_backing_dev_info')
> kernel/cgroup.c:227: error: storage size of 'cgroup_backing_dev_info' isn't known
> kernel/cgroup.c:227: warning: unused variable 'cgroup_backing_dev_info'
> make[1]: *** [kernel/cgroup.o] Error 1
> make: *** [kernel/cgroup.o] Error 2
>
>
> I haven't tested building other partial applications of the
> containersv11 patch set; any initial sequence of them should build.

This wasn't strictly the containersv11 patch set - it was a set of
patches from -mm that needed to be replaced in order to rename

containers to control groups. The initial subset of these patches was
the same as containersv11, but it also included a bunch of other
people's patches (containerstats, memory controller, etc).

>
> ... also ... too bad the names of these patches all have 'container' in
> them, not 'cgroup'. I realize how this came to be, due to changing
> container to cgroup after these patches were named, but it sure would
> be nice if the final patch set record that gets layed down in Linus's
> history showed the correct name of 'cgroup' in these patch names.

Agreed. Presumably that would be fairly straightforward for Andrew to
do directly on his quilt tree - I kept the names the same as the
previous patches to make it simpler to match up old to new.

Paul

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 11/29] task containersv11 make cpusets a client of containers
Posted by Paul Menage on Sun, 30 Sep 2007 07:11:18 GMT
View Forum Message <> Reply to Message

On 9/29/07, Paul Jackson <pj@sgi.com> wrote:
> There are some 21 mentions of 'manage_mutex' in the comments of
> kernel/cpuset.c remaining after this patch is applied, but no such
> mutex exists anymore.
>

Oops, sorry - fixing this is on my todo list.

Paul

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 11/29] task containersv11 make cpusets a client of containers
Posted by Paul Jackson on Sun, 30 Sep 2007 07:19:15 GMT
View Forum Message <> Reply to Message

> fixing this is on my todo list.

thanks!

--

I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.925.600.0401

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 01/29] task containersv11 basic task container framework
Posted by akpm on Sun, 30 Sep 2007 09:03:30 GMT
View Forum Message <> Reply to Message

On Sun, 30 Sep 2007 00:10:18 -0700 "Paul Menage" <menage@google.com> wrote:

> On 9/29/07, Paul Jackson <pj@sgi.com> wrote:
> > Paul M:
> >
> > This patch doesn't build for me in the following case. If I apply the
> > rest of the containersv11 patches, it builds, but if I happen to bisect
> > into this set of patches having applied only:
>
> These patches weren't the normal style of incremental patchset - they
> were direct substitutions of the previous patches in Andrew's -mm
> tree.
>
> Andrew's approach when encountering patches that are, e.g. missing
> includes on some architectures, is to add a *-fix.patch to the series;
> in this case it looks as though the fix for the missing ia64 include
> was added towards the end of the series rather than directly after the
> patch that needed it.

It makes my poor little life heaps easier if people can refer to patches
via their original Subject: or, better, their filename in the -mm patch
series.

I have vague memories that one of the container/cgroup fixup patches was a
bit of a jumbo patch which intersected multiple preceding patches. Often I
will refactor such a patch into several little *-fix.patch patches so that
everything lands nicely, but this time it was all too much effort and I
just jammmed <whichever patch it was> onto the tail of the queue.

And I think that's OK, because we don't care about git bisectability with
CONFIG_CGROUPS=y (I think?) Anyone who is bisecting through the middle of
the containers patch series is not searching for a containers problem, so

they can just configure it all off.

If we broke the build when CONFIG_CGROUPS=n then yeah, that's a problem.

wrt Paul's observation:

> ... also ... too bad the names of these patches all have 'container' in
> them, not 'cgroup'. I realize how this came to be, due to changing
> container to cgroup after these patches were named, but it sure would
> be nice if the final patch set record that gets layed down in Linus's
> history showed the correct name of 'cgroup' in these patch names.

that'll be OK. The "task-containers" text appears only in the filenmaes in the -mm patch series. Those filenames don't make an appearance in the git tree at all, so I didn't bother renaming all the files.

> Hence the fact that you couldn't compile until
> that later patch had been pushed.
>
> Andrew, is it practical for you to collapse any *-fix.patch files for
> cgroups into the appropriate patches that they fix? That would
> simplify the patch set a bit.

Yep, I always do that prior to sending to Linus, so everything hits the git tree as nicely as possible.

I normally defer that patch-folding exercise until the last minute, but I will occasionally do a big fold-some-patches-together pass, just to reduce the plain number of patches which I'm carrying. It gets irritating when this:

box:/usr/src/25> grep foo patches/*
zsh: argument list too long: grep

happens.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 01/29] task containersv11 basic task container framework
Posted by Paul Jackson on Sun, 30 Sep 2007 09:15:36 GMT

Andrew wrote:
> And I think that's OK, because we don't care about git bisectability with
> CONFIG_CGROUPS=y (I think?) Anyone who is bisecting through the middle of
> the containers patch series is not searching for a containers problem, so
> they can just configure it all off.

Ok - I guess.

In *-mm, this seems fine. I'd have figured that in Linus's tree, a wider
variety of people will be bisecting, some of them with CONFIG_CGROUPS=y
who don't even know what a CGROUP is, and so we'd want bisect to work
for them.

But you (Andrew) would know better than I.

> that'll be OK. The "task-containers" text appears only in the filenmaes in
> the -mm patch series. Those filenames don't make an appearance in the git
> tree at all, so I didn't bother renaming all the files.

Ah - ok - good.

--

I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.925.600.0401

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 01/29] task containersv11 basic task container framework
Posted by akpm on Sun, 30 Sep 2007 09:29:42 GMT

On Sun, 30 Sep 2007 02:15:36 -0700 Paul Jackson <pj@sgi.com> wrote:

> Andrew wrote:
> > And I think that's OK, because we don't care about git bisectability with
> > CONFIG_CGROUPS=y (I think?) Anyone who is bisecting through the middle of
> > the containers patch series is not searching for a containers problem, so
> > they can just configure it all off.
>
> Ok - I guess.
>
> In *-mm, this seems fine. I'd have figured that in Linus's tree, a wider

> variety of people will be bisecting, some of them with CONFIG_CGROUPS=y
> who don't even know what a CGROUP is, and so we'd want bisect to work
> for them.

Well obviously the situation is less than ideal, but it's livable with.

If one of their bisection points precedes the crgroups patch series,
their `make oldconfig' will knock out cgroups for them anyway, so when
their bisection cursor moves within or after the cgroups patches, they'll
still have CONFIG_CGROUPS=n. Serendipity.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 01/29] task containersv11 basic task container framework
Posted by Paul Jackson on Sun, 30 Sep 2007 09:36:03 GMT
View Forum Message <> Reply to Message

Andrew wrote:
> so when
> their bisection cursor moves within or after the cgroups patches, they'll
> still have CONFIG_CGROUPS=n. Serendipity.

Unless it's one of the configs that enables cgroups by default, thanks
to the patch:

  task-containers-enable-containers-by-default-in-some-configs.patch

namely configs:

 arch/ia64/configs/sn2_defconfig | 1 +
 arch/mips/configs/ip27_defconfig | 1 +
 arch/mips/configs/sb1250-swarm_defconfig | 1 +
 arch/powerpc/configs/cell_defconfig | 1 +
 arch/powerpc/configs/ppc64_defconfig | 1 +
 arch/powerpc/configs/pseries_defconfig | 1 +

But these are not commonly used configs, so that's not a big deal.

So ... yeah ... good enough.

--
                  I won't rest till it's the best ...
                  Programmer, Linux Scalability
                  Paul Jackson <pj@sgi.com> 1.925.600.0401

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers