Subject: Re: How to draw values for /proc/stat
Posted by Glauber Costa on Sun, 11 Dec 2011 14:50:56 GMT
View Forum Message <> Reply to Message

On 12/09/2011 03:55 PM, Glauber Costa wrote:
> On 12/09/2011 12:03 PM, Peter Zijlstra wrote:
>> On Mon, 2011-12-05 at 07:32 -0200, Glauber Costa wrote:
>>> Hi,
>>>
>>> Specially Peter and Paul, but all the others:
>>>
>>> As you can see in https://lkml.org/lkml/2011/12/4/178, and in my answer
>>> to that, there is a question - one I've asked before but without that
>>> much of an audience - of whether /proc files read from process living on
>>> cgroups should display global or per-cgroup resources.
>>>
>>> In the past, I was arguing for a knob to control that, but I recently
>>> started to believe that a knob here will only overcomplicate matters:
>>> if you live in a cgroup, you should display only the resources you can
>>> possibly use. Global is for whoever is in the main cgroup.
>>>
>>> Now, it comes two questions:
>>> 1) Do you agree with that, for files like /proc/stat ? I think the most
>>> important part is to be consistent inside the system, regardless of what
>>> is done
>>
>> Personally I don't give a rats arse about (/proc vs) cgroups :-)
>> Currently /proc is unaffected by whatever cgroup you happen to be in and
>> that seems to make some sort of sense.
>>
>> Namespaces seem to be about limiting visibility, cgroups about
>> controlling resources.
>>
>> The two things are hopelessly disjoint atm, but I believe someone was
>> looking at this mess.
>
> I did take a look at this (if anyone else was, I'd like to know so we
> can share some ideas), but I am not convinced we should do anything to
> join them anymore. We virtualization people are to the best of my
> knowledge the only ones doing namespaces. Cgroups, OTOH, got a lot bigger.
>
> What I am mostly concerned about now, is how consistent they will be.
> /proc always being always global indeed does make sense, but my question
> still stands: if you live in a resource-controlled world, why should you
> even see resources you will never own ?
>
>
>> IOW a /proc namespace coupled to cgroup scope would do what you want.

>> Now my head hurts..
>
> Mine too. The idea is good, but too broad. Boils down to: How do you
> couple them? And none of the methods I thought about seemed to make any
> sense.
>
> If we really want to have the values in /proc being opted-in, I think
> Kamezawa's idea of a mount option is the winner so far.
>

Ok:

How about the following patch to achieve this ?

From 1e587bf3d97d850c5ba8b9bf375c2e74b38a9891 Mon Sep 17 00:00:00 2001
From: Glauber Costa <glommer@parallels.com>
Date: Fri, 9 Dec 2011 16:03:26 +0300
Subject: [PATCH] Add "proc_overlay" option for cgroup

This patch adds the "proc_overlay" file to cgroup. It is also
available as a mount option, which can be overriden later by
any of the individual cgroups in the hierarchy.

With this option set, we tell the kernel that for the processes
inside cgroups for which proc_overlay = true, we want /proc files
related to this cgroup to show per-cgroup values, not global ones.

This is a must-have in virtualized environments, where the isolation
guarantees should not let processes in a cgroup see resources of
other groups. Adding an option allow us to achieve this without
interferring with other cgroup-users that are not concerned with
isolation.

Signed-off-by: Glauber Costa <glommer@parallels.com>
---
 include/linux/cgroup.h | 6 ++++++
 kernel/cgroup.c | 37 +++++++++++++++++++++++++++++++++++++
 2 files changed, 43 insertions(+), 0 deletions(-)

diff --git a/include/linux/cgroup.h b/include/linux/cgroup.h
index 1b7f9d5..f0bc2e9 100644
--- a/include/linux/cgroup.h
+++ b/include/linux/cgroup.h
@@ -158,6 +158,7 @@ enum {
   * Clone cgroup values when creating a new child cgroup
   */
  CGRP_CLONE_CHILDREN,
+ CGRP_PROC_OVERLAY,

```
 };

 /* which pidlist file are we talking about? */
@@ -245,6 +246,11 @@ struct cgroup {
  spinlock_t event_list_lock;
 };

+static inline bool cgroup_proc_overlay(struct cgroup *cgrp)
+{
+ return test_bit(CGRP_PROC_OVERLAY, &cgrp->flags);
+}
+
 /*
  * A css_set is a structure holding pointers to a set of
  * cgroup_subsys_state objects. This saves space in the task struct
diff --git a/kernel/cgroup.c b/kernel/cgroup.c
index e700abe..313c06c 100644
--- a/kernel/cgroup.c
+++ b/kernel/cgroup.c
@@ -233,6 +233,7 @@ enum {
  ROOT_NOPREFIX, /* mounted subsystems have no named prefix */
  ROOT_CLONE_CHILDREN, /* mounted subsystems starts with clone_children */
  ROOT_NOSUBSYS, /* explicitly asked for 'none' subsystems */
+ ROOT_PROC_OVERLAY, /* mounted subsystems starts with proc_overlay */
 };

 static int cgroup_is_releasable(const struct cgroup *cgrp)
@@ -253,6 +254,10 @@ static inline int clone_children(const struct cgroup *cgrp)
  return test_bit(CGRP_CLONE_CHILDREN, &cgrp->flags);
 }

+static inline int proc_overlay(const struct cgroup *cgrp)
+{
+ return test_bit(CGRP_PROC_OVERLAY, &cgrp->flags);
+}
 /*
  * for_each_subsys() allows you to iterate on each subsystem attached to
  * an active hierarchy
@@ -1054,6 +1059,8 @@ static int cgroup_show_options(struct seq_file *seq, struct vfsmount
*vfs)
   seq_printf(seq, ",release_agent=%s", root->release_agent_path);
  if (clone_children(&root->top_cgroup))
   seq_puts(seq, ",clone_children");
+ if (test_bit(ROOT_PROC_OVERLAY, &root->flags))
+  seq_puts(seq, ",proc_overlay");
  if (strlen(root->name))
   seq_printf(seq, ",name=%s", root->name);
  mutex_unlock(&cgroup_mutex);
```

```
@@ -1116,6 +1123,10 @@ static int parse_cgroupfs_options(char *data, struct cgroup_sb_opts
*opts)
    set_bit(ROOT_NOPREFIX, &opts->flags);
    continue;
   }
+  if (!strcmp(token, "proc_overlay")) {
+   set_bit(ROOT_PROC_OVERLAY, &opts->flags);
+   continue;
+  }
   if (!strcmp(token, "clone_children")) {
    set_bit(ROOT_CLONE_CHILDREN, &opts->flags);
    continue;
@@ -1406,6 +1417,8 @@ static struct cgroupfs_root *cgroup_root_from_opts(struct
cgroup_sb_opts *opts)
   strcpy(root->name, opts->name);
  if (test_bit(ROOT_CLONE_CHILDREN, &opts->flags))
   set_bit(CGRP_CLONE_CHILDREN, &root->top_cgroup.flags);
+ if (test_bit(ROOT_PROC_OVERLAY, &opts->flags))
+  set_bit(CGRP_PROC_OVERLAY, &root->top_cgroup.flags);
  return root;
 }

@@ -3442,6 +3455,22 @@ static int cgroup_write_notify_on_release(struct cgroup *cgrp,
  return 0;
 }

+static u64 cgroup_proc_overlay_read(struct cgroup *cgrp,
+        struct cftype *cft)
+{
+ return proc_overlay(cgrp);
+}
+
+static int cgroup_proc_overlay_write(struct cgroup *cgrp,
+         struct cftype *cft, u64 val)
+{
+ if (val)
+  set_bit(CGRP_PROC_OVERLAY, &cgrp->flags);
+ else
+  clear_bit(CGRP_PROC_OVERLAY, &cgrp->flags);
+ return 0;
+}
+
 /*
  * Unregister event and free resources.
  *
@@ -3662,6 +3691,11 @@ static struct cftype files[] = {
   .read_u64 = cgroup_clone_children_read,
   .write_u64 = cgroup_clone_children_write,
```

```
  },
+ {
+  .name = "cgroup.proc_overlay",
+  .read_u64 = cgroup_proc_overlay_read,
+  .write_u64 = cgroup_proc_overlay_write,
+ },
 };

 static struct cftype cft_release_agent = {
@@ -3794,6 +3828,9 @@ static long cgroup_create(struct cgroup *parent, struct dentry *dentry,
  if (clone_children(parent))
   set_bit(CGRP_CLONE_CHILDREN, &cgrp->flags);

+ if (parent->root->flags & ROOT_PROC_OVERLAY)
+  set_bit(CGRP_PROC_OVERLAY, &cgrp->flags);
+
  for_each_subsys(root, ss) {
   struct cgroup_subsys_state *css = ss->create(ss, cgrp);

--
1.7.6.4
```

File Attachments

1) 0001-Add-proc_overlay-option-for-cgroup.patch, downloaded 1482 times