Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by Paul Menage on Wed, 14 May 2008 08:44:38 GMT
View Forum Message <> Reply to Message

On Tue, May 13, 2008 at 8:21 PM, YAMAMOTO Takashi
<yamamoto@valinux.co.jp> wrote:
> >
> > Could you please mention what the limitations are? We could get those fixed or
> > take another serious look at the mm->owner patches.
>
> for example, its callback can't sleep.
>

You need to be able to sleep in order to take mmap_sem, right?
Since I think that the other current user of the mm->owner callback
probably also needs mmap_sem, it might make sense to take mmap_sem in
mm_update_next_owner() prior to locking the old owner, and hold it
across the callback, which would presumably solve the problem.

> >
> > Isn't it bad to force a group to go over it's limit due to migration?
>
> we don't have many choices as far as ->attach can't fail.
> although we can have racy checks in ->can_attach, i'm not happy with it.

can_attach() isn't racy iff you ensure that a successful result from
can_attach() can't be invalidated by any code not holding
cgroup_mutex.

The existing user of can_attach() is cpusets, and the only way to make
an attachable cpuset non-attachable is to remove its last node or cpu.
The only code that can do this (update_nodemask, update_cpumask, and
common_cpu_mem_hotplug_unplug) all call cgroup_lock() to ensure that
this synchronization occurs.

Of course, having lots of datapath operations also take cgroup_mutex
would be really painful, so it's not practical to use for things that
can become non-attachable due to a process consuming some resources.
This is part of the reason that I started working on the lock-mode
patches that I sent out yesterday, in order to make finer-grained
locking simpler. I'm going to rework those to make the locking more
explicit, and I'll bear this use case in mind while I'm doing it.

A few comments on the patch:

- you're not really limiting swap usage, you're limiting swapped-out
address space. So it looks as though if a process has swapped out most
of its address space, and forks a child, the total "swap" charge for

the cgroup will double. Is that correct? If so, why is this better
than charging for actual swap usage?

- what will happen if someone creates non-NPTL threads, which share an
mm but not a thread group (so each of them is a thread group leader)?

- if you were to store a pointer in the page rather than the
swap_cgroup pointer, then (in combination with mm->owner) you wouldn't
need to do the rebinding to the new swap_cgroup when a process moves
to a different cgroup - you could instead keep a "swapped pte" count
in the mm, and just charge that to the new cgroup and uncharge it from
the old cgroup. You also wouldn't need to keep ref counts on the
swap_cgroup.

- ideally this wouldn't actually start charging until it was bound on
to a cgroups hierarchy, although I guess that the performance of this
is less important than something like the virtual address space
controller, since once we start swapping we can expect performance to
be bad anyway.

Paul

_____