Subject: Re: [RFC][PATCH 3/7] Data structures changes for RSS accounting
Posted by dev on Mon, 12 Mar 2007 16:16:59 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Pavel Emelianov <xemul@sw.ru> writes:
>
>
>>Adds needed pointers to mm_struct and page struct,
>>places hooks to core code for mm_struct initialization
>>and hooks in container_init_early() to preinitialize
>>RSS accounting subsystem.
>
>
> An extra pointer in struct page is unlikely to fly.
> Both because it increases the size of a size critical structure,
> and because conceptually it is ridiculous.
as it was discussed multiple times (and according OLS):
- it is not critical nowdays to expand struct page a bit in case
  accounting is on.
- it can be done w/o extending, e.g. via mapping page <-> container
  using hash or some other data structure.
  i.e. we can optimize it on size if considered needed.

> If you are limiting the RSS size you are counting the number of pages in
> the page tables. You don't care about the page itself.
>
> With the rmap code it is relatively straight forward to see if this is
> the first time a page has been added to a page table in your rss
> group, or if this is the last reference to a particular page in your
> rss group. The counters should only increment the first time a
> particular page is added to your rss group. The counters should only
> decrement when it is the last reference in your rss subsystem.
You are fundamentally wrong if shared pages are concerned.
Imagine a glibc page shared between 2 containers - VE1 and VE2.
VE1 was the first who mapped it, so it is accounted to VE1
(rmap count was increased by it).
now VE2 maps the same page. You can't determine whether this page is mapped
to this container or another one w/o page->container pointer.
All the choices you have are:
a) do not account this page, since it is allready accounted to some other VE.
b) account this page again to current container.

(a) is bad, since VE1 can unmap this page first, and the last user will be VE2.
Which means VE1 will be charged for it, while VE2 uncharged. Accounting screws up.

b) is bad, since:
  - the same page is accounted multiple times, which makes impossible

   to understand how much real memory pages container needs/consumes
  - and because on container enter the process and it's pages
   are essentially moved to another context, while accounting
   can not be fixed up easily and we essentially have (a).

> This allow important little cases like glibc to be properly accounted
> for. One of the key features of a rss limit is that the kernel can
> still keep pages that you need in-core, that are accessible with just
> a minor fault. Directly owning pages works directly against that
> principle.

Sorry, can't understand what you mean. It doesn't work against.
Each container has it's own LRU. So if glibc has the most
often used pages - it won't be thrashed out.

Thanks,
Kirill