

---

Subject: Re: [PATCH 0/8] RSS controller based on process containers (v3.1)

Posted by [Herbert Poetzl](#) on Fri, 08 Jun 2007 12:03:43 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

On Mon, Jun 04, 2007 at 05:25:25PM +0400, Pavel Emelianov wrote:

> Adds RSS accounting and control within a container.

>

> Changes from v3

> - comments across the code

> - git-bisect safe split

> - lost places to move the page between active/inactive lists

>

> Ported above Paul's containers V10 with fixes from Balbir.

>

> RSS container includes the per-container RSS accounting

> and reclamation, and out-of-memory killer.

>

>

> Each mapped page has an owning container and is linked into its

> LRU lists just like in the global LRU ones. The owner of the page

> is the container that touched the page first.

> As long as the page stays mapped it holds the container, is accounted

> into its usage and lives in its LRU list. When page is unmapped for

> the last time it releases the container.

> The RSS usage is exactly the number of pages in its booth LRU lists,

> i.e. the number of pages used by this container.

so there could be two guests, unified (i.e. sharing most of the files as hardlinks), where the first one holds 80% of the resulting pages, and the second one 20%, and thus shows much lower 'RSS' usage as the other one, although it is running the very same processes and providing identical services?

> When this usage exceeds the limit set some pages are reclaimed from

> the owning container. In case no reclamation possible the OOM killer

> starts thinning out the container.

so the system (physical machine) starts reclaiming and probably swapping even when there is no need to do so?

e.g. a system with a single guest, limited to 10k pages, with a working set of 15k pages in different apps would continuously swap (trash?) on an otherwise unused (100k+ pages) system?

> Thus the container behaves like a standalone machine - when it runs  
> out of resources, it tries to reclaim some pages, and if it doesn't  
> succeed, kills some task.

is that really what we want?

I think we can do better than a standalone machine  
and in many cases we really should ...

best,  
Herbert

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

>

> The testing scenario may look like this:

>

> 1. Prepare the containers

> # mkdir -p /containers/rss

> # mount -t container none /containers/rss -o rss

>

> 2. Make the new group and move bash into it

> # mkdir /containers/rss/0

> # echo \$\$ > /containers/rss/0/tasks

>

> Since now we're in the 0 container.

> We can alter the RSS limit

> # echo -n 6000 > /containers/rss/0/rss\_limit

>

> We can check the usage

> # cat /containers/rss/0/rss\_usage

> 25

>

> And do other stuff. To check the reclamation to work we need a  
> simple program that touches many pages of memory, like this:

>

> #include <stdio.h>

> #include <unistd.h>

> #include <sys/mman.h>

> #include <fcntl.h>

>

> #ifndef PGSIZE

> #define PGSIZE 4096

> #endif

>

> int main(int argc, char \*\*argv)

> {

> unsigned long pages;

> int i;

```

> char *mem;
>
> if (argc < 2) {
>     printf("Usage: %s <number_of_pages>\n", argv[0]);
>     return 1;
> }
>
> pages = strtol(argv[1], &mem, 10);
> if (*mem != '\0') {
>     printf("Bad number %s\n", argv[1]);
>     return 1;
> }
>
> mem = mmap(NULL, pages * PGSIZE, PROT_READ | PROT_WRITE,
>     MAP_PRIVATE | MAP_ANON, 0, 0);
> if (mem == MAP_FAILED) {
>     perror("map");
>     return 2;
> }
>
> for (i = 0; i < pages; i++)
>     mem[i * PGSIZE] = 0;
>
> printf("OK\n");
> return 0;
> }

```

---

```

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

```

---

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

---