Subject: Re: [RFC][PATCH][1/4] RSS controller setup
Posted by Balbir Singh on Mon, 19 Feb 2007 10:06:49 GMT
View Forum Message <> Reply to Message

Andrew Morton wrote:
> On Mon, 19 Feb 2007 12:20:26 +0530 Balbir Singh <balbir@in.ibm.com> wrote:
>
>> This patch sets up the basic controller infrastructure on top of the
>> containers infrastructure. Two files are provided for monitoring
>> and control memctlr_usage and memctlr_limit.
>
> The patches use the identifier "memctlr" a lot. It is hard to remember,
> and unpronounceable. Something like memcontrol or mem_controller or
> memory_controller would be more typical.
>

I'll change the name to memory_controller

```
>> ...
>>
>> + BUG_ON(!mem);
>> + if ((buffer = kmalloc(nbytes + 1, GFP_KERNEL)) == 0)
>> +  return -ENOMEM;
>
> Please prefer to do
>
>  buffer = kmalloc(nbytes + 1, GFP_KERNEL);
>  if (buffer == NULL)
>   reutrn -ENOMEM;
>
> ie: avoid the assign-and-test-in-the-same-statement thing.  This affects
> the whole patchset.
>
```

I'll fix that

> Also, please don't compare pointers to literal zero like that. It makes me
> get buried it patches to convert it to "NULL". I think this is a sparse
> thing.
>

Good point, I'll fix it.

```
>> + buffer[nbytes] = 0;
>> + if (copy_from_user(buffer, userbuf, nbytes)) {
>> +  ret = -EFAULT;
>> +  goto out_err;
>> + }
>> +
>> + container_manage_lock();
>> + if (container_is_removed(cont)) {
>> +  ret = -ENODEV;
>> +  goto out_unlock;
>> + }
>> +
>> + limit = simple_strtoul(buffer, NULL, 10);
>> + /*
>> +  * 0 is a valid limit (unlimited resource usage)
>> +  */
>> + if (!limit && strcmp(buffer, "0"))
>> +  goto out_unlock;
>> +
>> + spin_lock(&mem->lock);
>> + mem->counter.limit = limit;
>> + spin_unlock(&mem->lock);
>
> The patches do this a lot: a single atomic assignment with a
> pointless-looking lock/unlock around it.  It's often the case that this
> idiom indicates a bug, or needless locking.  I think the only case where it
> makes sense is when there's some other code somewhere which is doing
>
>  spin_lock(&mem->lock);
>  ...
>  use1(mem->counter.limit);
>  ...
>  use2(mem->counter.limit);
>  ...
>  spin_unlock(&mem->lock);
>
> where use1() and use2() expect the two reads of mem->counter.limit to
> return the same value.
>
> Is that the case in these patches?  If not, we might have a problem in
> there.
>
```

The next set of patches move to atomic values for the limits. That should fix the locking.

```
>> +
>> +static ssize_t memctlr_read(struct container *cont, struct cftype *cft,
>> +    struct file *file, char __user *userbuf,
>> +    size_t nbytes, loff_t *ppos)
>> +{
>> + unsigned long usage, limit;
>> + char usagebuf[64];  /* Move away from stack later */
>> + char *s = usagebuf;
>> + struct memctlr *mem = memctlr_from_cont(cont);
>> +
>> + spin_lock(&mem->lock);
>> + usage = mem->counter.usage;
>> + limit = mem->counter.limit;
>> + spin_unlock(&mem->lock);
>> +
>> + s += sprintf(s, "usage %lu, limit %ld\n", usage, limit);
>> + return simple_read_from_buffer(userbuf, nbytes, ppos, usagebuf,
>> +     s - usagebuf);
>> +}
>
> This output is hard to parse and to extend.  I'd suggest either two
> separate files, or multi-line output:
>
> usage: %lu kB
> limit: %lu kB
>
> and what are the units of these numbers?  Page counts?  If so, please don't
> do that: it requires appplications and humans to know the current kernel's
> page size.
>
```

Yes, this looks much better. I'll move to this format. I get myself lost
in "bc" at times, that should have been a hint.

```
>> +static struct cftype memctlr_usage = {
>> + .name = "memctlr_usage",
>> + .read = memctlr_read,
>> +};
>> +
>> +static struct cftype memctlr_limit = {
>> + .name = "memctlr_limit",
>> + .write = memctlr_write,
>> +};
>> +
>> +static int memctlr_populate(struct container_subsys *ss,
>> +    struct container *cont)
>> +{
>> + int rc;
>> + if ((rc = container_add_file(cont, &memctlr_usage)) < 0)
>> +  return rc;
>> + if ((rc = container_add_file(cont, &memctlr_limit)) < 0)
>
> Clean up the first file here?
>
```

I used cpuset_populate() as an example to code this one up.
I don't think there is an easy way in containers to clean up
files. I'll double check

```
>> +  return rc;
>> + return 0;
>> +}
>> +
>> +static struct container_subsys memctlr_subsys = {
>> + .name = "memctlr",
>> + .create = memctlr_create,
>> + .destroy = memctlr_destroy,
>> + .populate = memctlr_populate,
>> +};
>> +
>> +int __init memctlr_init(void)
>> +{
>> + int id;
>> +
>> + id = container_register_subsys(&memctlr_subsys);
>> + printk("Initializing memctlr version %s, id %d\n", version, id);
>> + return id < 0 ? id : 0;
>> +}
>> +
>> +module_init(memctlr_init);
>
```

Thanks for the detailed review,

--
 Warm Regards,
 Balbir Singh