Subject: Re: [-mm PATCH 1/8] Memory controller resource counters (v2)
Posted by Balbir Singh on Fri, 06 Jul 2007 21:03:58 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Thu, 2007-07-05 at 22:20 -0700, Balbir Singh wrote:
>> +/*
>> + * the core object. the container that wishes to account for some
>> + * resource may include this counter into its structures and use
>> + * the helpers described beyond
>> + */
>
> I'm going to nitpick a bit here. Nothing major, I promise. ;)
>
> Could we make these comments into nice sentences with capitalization? I
> think it makes them easier to read in long comments.
>
> How about something like this for the comment:
>
> /*
> * A container wishing to account for a resource should include this
> * structure into one of its own. It may use the helpers below.
> */
>
> The one above is worded a little bit strangely.
>

Hi, Dave,

These patches were posted by Pavel, I've carried them forward as is.
Suggestions are always welcome.

>> +struct res_counter {
>> + /*
>> + * the current resource consumption level
>> + */
>> + unsigned long usage;
>> + /*
>> + * the limit that usage cannot exceed
>> + */
>> + unsigned long limit;
>> + /*
>> + * the number of insuccessful attempts to consume the resource
>> + */
>
> unsuccessful
>

Thanks, fixed.

>> + unsigned long failcnt;
>> + /*
>> + * the lock to protect all of the above.
>> + * the routines below consider this to be IRQ-safe
>> + */
>> + spinlock_t lock;
>> +};
>
> Do we really need all of these comments? Some of them are a wee bit
> self-explanatory. I think we mostly know what a limit is. ;)
>

I'll leave the decision on the comments exclusion to Pavel.

>
> More nitpicking...
>
> Can we leave the normal control flow in the lowest indentation level,
> and have only errors in the indented if(){} blocks? Something like
> this:
>

Sounds good, done!

>> +int res_counter_charge_locked(struct res_counter *cnt, unsigned long
> val)
>> +{
>> + if (cnt->usage > cnt->limit - val) {
>> + cnt->failcnt++;
>> + return -ENOMEM;
>> + }
>> + cnt->usage += val;
>> + return 0;
>> +}
>
> Also, can you do my poor brain a favor an expand "cnt" to "counter"?
> You're not saving _that_ much typing ;)
>

Done

>> +int res_counter_charge(struct res_counter *cnt, unsigned long val)
>> +{
>> + int ret;
>> + unsigned long flags;

```
>> +
>> + spin_lock_irqsave(&cnt->lock, flags);
>> + ret = res_counter_charge_locked(cnt, val);
>> + spin_unlock_irqrestore(&cnt->lock, flags);
>> + return ret;
>> +}
>> +
>> +void res_counter_uncharge_locked(struct res_counter *cnt, unsigned long val)
>> +{
>> + if (unlikely(cnt->usage < val)) {
>> +  WARN_ON(1);
>> +  val = cnt->usage;
>> + }
>> +
>> + cnt->usage -= val;
>> +}
>
> It actually looks like the WARN_ON() macros "return" values.  You should
> be able to:
>
>  if (WARN_ON(cnt->usage < val))
>   val = count->usage;
>
```

I think, thats better, will change it

```
>> +void res_counter_uncharge(struct res_counter *cnt, unsigned long val)
>> +{
>> + unsigned long flags;
>> +
>> + spin_lock_irqsave(&cnt->lock, flags);
>> + res_counter_uncharge_locked(cnt, val);
>> + spin_unlock_irqrestore(&cnt->lock, flags);
>> +}
>> +
>> +
>> +static inline unsigned long *res_counter_member(struct res_counter *cnt, int member)
>> +{
>> + switch (member) {
>> + case RES_USAGE:
>> +  return &cnt->usage;
>> + case RES_LIMIT:
>> +  return &cnt->limit;
>> + case RES_FAILCNT:
>> +  return &cnt->failcnt;
>> + };
>> +
>> + BUG();
```

>> + return NULL;
>> +}
>>
>> +ssize_t res_counter_read(struct res_counter *cnt, int member,
>> + const char __user *userbuf, size_t nbytes, loff_t *pos)
>> +{
>> + unsigned long *val;
>> + char buf[64], *s;
>> +
>> + s = buf;
>> + val = res_counter_member(cnt, member);
>> + s += sprintf(s, "%lu\n", *val);
>> + return simple_read_from_buffer((void __user *)userbuf, nbytes,
>> +   pos, buf, s - buf);
>> +}
>
> Why do we need that cast?
>

u mean the __user? If I remember correctly it's a attribute for sparse.

>> +ssize_t res_counter_write(struct res_counter *cnt, int member,
>> + const char __user *userbuf, size_t nbytes, loff_t *pos)
>> +{
>> + int ret;
>> + char *buf, *end;
>> + unsigned long tmp, *val;
>> +
>> + buf = kmalloc(nbytes + 1, GFP_KERNEL);
>
> Do we need some checking on nbytes? Is it sanitized before it gets
> here?
>

I think the container infrastructure should handle that.

>> + ret = -ENOMEM;
>> + if (buf == NULL)
>> + goto out;
>> +
>> + buf[nbytes] = 0;
>
> Please use '\0'. 0 isn't a char.
>

Yep, will do.

>> + ret = -EFAULT;

>> + if (copy_from_user(buf, userbuf, nbytes))
>> + goto out_free;
>> +
>> + ret = -EINVAL;
>> + tmp = simple_strtoul(buf, &end, 10);
>> + if (*end != '\0')
>> + goto out_free;
>> +
>> + val = res_counter_member(cnt, member);
>> + *val = tmp;
>> + ret = nbytes;
>> +out_free:
>> + kfree(buf);
>> +out:
>> + return ret;
>> +}
>> _
>>
> -- Dave
>

--
Thanks,
Balbir Singh
Linux Technology Center
IBM, ISTL

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