Subject: Re: [-mm PATCH 1/8] Memory controller resource counters (v2)
Posted by Dave Hansen on Mon, 09 Jul 2007 19:56:48 GMT
View Forum Message <> Reply to Message

```
On Mon, 2007-07-09 at 11:16 +0400, Pavel Emelianov wrote:
> 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.
> > 
> >> +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
> > 
> >> + 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. ;)
>
> Since this is a new entities in the kernel and not many people
> deal with the resource management, I think that nothing bad in
> having them.

They waste space. It makes the code harder to read.

```
> >> +/*
> >> + * helpers to interact with userspace
> >> + * res_counter_read/_write - put/get the specified fields from the
> >> + * res_counter struct to/from the user
> >> + *
> >> + * @cnt:     the counter in question
> >> + * @member:  the field to work with (see RES_xxx below)
> >> + * @buf:     the buffer to opeate on,...
> >> + * @nbytes:  its size...
> >> + * @pos:     and the offset.
> >> + */
> >> +
> >> +ssize_t res_counter_read(struct res_counter *cnt, int member,
> >> +  const char __user *buf, size_t nbytes, loff_t *pos);
> >> +ssize_t res_counter_write(struct res_counter *cnt, int member,
> >> +  const char __user *buf, size_t nbytes, loff_t *pos);
> >> +
> >> +/*
> >> + * the field descriptors. one for each member of res_counter
> >> + */
> >> +
> >> +enum {
> >> + RES_USAGE,
> >> + RES_LIMIT,
> >> + RES_FAILCNT,
> >> +};
> >> +
>
> [snip]
>
> >> diff -puN /dev/null kernel/res_counter.c
> >> --- /dev/null 2007-06-01 08:12:04.000000000 -0700
> >> +++ linux-2.6.22-rc6-balbir/kernel/res_counter.c 2007-07-05 13:45:17.000000000 -0700
> >> @@ -0,0 +1,121 @@
> >> +/*
> >> + * resource containers
> >> + *
> >> + * Copyright 2007 OpenVZ SWsoft Inc
> >> + *
> >> + * Author: Pavel Emelianov <xemul@openvz.org>
> >> + *
> >> + */
> >> +
> >> +#include <linux/types.h>
> >> +#include <linux/parser.h>
> >> +#include <linux/fs.h>
> >> +#include <linux/res_counter.h>
> >> +#include <linux/uaccess.h>
> >> +
> >> +void res_counter_init(struct res_counter *cnt)
> >> +{
> >> + spin_lock_init(&cnt->lock);
> >> + cnt->limit = (unsigned long)LONG_MAX;
> >> +}
> >> +
> >> +int res_counter_charge_locked(struct res_counter *cnt, unsigned long val)
> >> +{
> >> + if (cnt->usage <= cnt->limit - val) {
> >> +  cnt->usage += val;
> >> +  return 0;
> >> + }
> >> +
> >> + cnt->failcnt++;
> >> + return -ENOMEM;
> >> +}
```
> >
> > 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:
>
> As far as I know gcc usually makes the "true" branch to be
> in the straight code flow and in general case this does not
> trash the CPU pipeline.

It's not a big deal either way, but that's a pretty weak reason for
doing it that way. Can you actually demonstrate a performance
difference? If not, we should defer to the most readable form.

```
> >> +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?
>
> simple_read_from_buffer do not take const char * as the 1st arg
```

True, but we can pass char* to a function taking void* without a problem
and without an explicit cast.

What's the actual problem? The "const"? We're effectively throwing
away the information here that res_counter_read() expects userbuf to be
constant. If simple_read_from_buffer() ever decided to write to
userbuf, we'd be in trouble. If simple_read_from_buffer() will never
write, then _it_ should have a const first argument.

Also, what if "userbuf" changes type? We'll never see warnings, just
weird runtime bugs.

I just worry that these kinds of casts shut up warnings that _are_ valid
and might find real bugs.

-- Dave