

---

Subject: Re: [RFC][PATCH 1/7] Resource counters  
Posted by [xemul](#) on Wed, 07 Mar 2007 07:17:00 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Balbir Singh wrote:

> Pavel Emelianov wrote:

>> Introduce generic structures and routines for  
>> resource accounting.

>>

>> Each resource accounting container is supposed to  
>> aggregate it, container\_subsystem\_state and its  
>> resource-specific members within.

>>

>>

>> -----

>>

>> diff -upr linux-2.6.20.orig/include/linux/res\_counter.h

>> linux-2.6.20-0/include/linux/res\_counter.h

>> --- linux-2.6.20.orig/include/linux/res\_counter.h 2007-03-06

>> 13:39:17.000000000 +0300

>> +++ linux-2.6.20-0/include/linux/res\_counter.h 2007-03-06

>> 13:33:28.000000000 +0300

>> @@ -0,0 +1,83 @@

>> + #ifndef \_\_RES\_COUNTER\_H\_\_

>> + #define \_\_RES\_COUNTER\_H\_\_

>> + /\*

>> + \* resource counters

>> + \*

>> + \* Copyright 2007 OpenVZ SWsoft Inc

>> + \*

>> + \* Author: Pavel Emelianov <xemul@openvz.org>

>> + \*

>> + \*/

>> +

>> + #include <linux/container.h>

>> +

>> + struct res\_counter {

>> + unsigned long usage;

>> + unsigned long limit;

>> + unsigned long failcnt;

>> + spinlock\_t lock;

>> +};

>> +

>> + enum {

>> + RES\_USAGE,

>> + RES\_LIMIT,

>> + RES\_FAILCNT,

>> +};

```

>> +
>> +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);
>> +
>> +static inline void res_counter_init(struct res_counter *cnt)
>> +{
>> +    spin_lock_init(&cnt->lock);
>> +    cnt->limit = (unsigned long)LONG_MAX;
>> +}
>> +
>
> Is there any way to indicate that there are no limits on this container.

```

Yes - LONG\_MAX is essentially a "no limit" value as no container will ever have such many files :)

> LONG\_MAX is quite huge, but still when the administrator wants to  
> configure a container to \*un-limited usage\*, it becomes hard for  
> the administrator.

```

>
>> +static inline 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;
>> +}
>> +
>> +static inline 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;
>> +}
>> +
>
> Will atomic counters help here.

```

I'm afraid no. We have to atomically check for limit and alter one of usage or failcnt depending on the checking result. Making this with atomic\_xxx ops will require at least two ops.

If we'll remove failcnt this would look like

```
while (atomic_cmpxchg(...))  
which is also not that good.
```

Moreover - in RSS accounting patches I perform page list manipulations under this lock, so this also saves one atomic op.

```
>> +static inline 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;  
>> +}  
>> +  
>> +static inline 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);  
>> +}  
>> +  
>> +#endif  
>> diff -upr linux-2.6.20.orig/init/Kconfig linux-2.6.20-0/init/Kconfig  
>> --- linux-2.6.20.orig/init/Kconfig 2007-03-06 13:33:28.000000000 +0300  
>> +++ linux-2.6.20-0/init/Kconfig 2007-03-06 13:33:28.000000000 +0300  
>> @@ -265,6 +265,10 @@ config CPUSETS  
>>  
>>     Say N if unsure.  
>>  
>> +config RESOURCE_COUNTERS  
>> +  bool  
>> +  select CONTAINERS  
>> +  
>> config SYSFS_DEPRECATED  
>>     bool "Create deprecated sysfs files"  
>>     default y
```

```

>> diff -upr linux-2.6.20.orig/kernel/Makefile
>> linux-2.6.20-0/kernel/Makefile
>> --- linux-2.6.20.orig/kernel/Makefile    2007-03-06 13:33:28.000000000
>> +0300
>> +++ linux-2.6.20-0/kernel/Makefile    2007-03-06 13:33:28.000000000 +0300
>> @@ -51,6 +51,7 @@ obj-$(CONFIG_RELAY) += relay.o
>> obj-$(CONFIG_UTS_NS) += utsname.o
>> obj-$(CONFIG_TASK_DELAY_ACCT) += delayacct.o
>> obj-$(CONFIG_TASKSTATS) += taskstats.o tsacct.o
>> +obj-$(CONFIG_RESOURCE_COUNTERS) += res_counter.o
>>
>> ifneq ($(CONFIG_SCHED_NO_NO_OMIT_FRAME_POINTER),y)
>> # According to Alan Modra <alan@linuxcare.com.au>, the
>> -fno-omit-frame-pointer is
>> diff -upr linux-2.6.20.orig/kernel/res_counter.c
>> linux-2.6.20-0/kernel/res_counter.c
>> --- linux-2.6.20.orig/kernel/res_counter.c    2007-03-06
>> 13:39:17.000000000 +0300
>> +++ linux-2.6.20-0/kernel/res_counter.c    2007-03-06
>> 13:33:28.000000000 +0300
>> @@ -0,0 +1,72 @@
>> +/*
>> + * resource containers
>> + *
>> + * Copyright 2007 OpenVZ SWsoft Inc
>> + *
>> + * Author: Pavel Emelianov <xemul@openvz.org>
>> + *
>> + */
>> +
>> +#include <linux/parser.h>
>> +#include <linux/fs.h>
>> +#include <linux/res_counter.h>
>> +#include <asm/uaccess.h>
>> +
>> +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);
>> +}
>> +
>> +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);
>> +    ret = -ENOMEM;
>> +    if (buf == NULL)
>> +        goto out;
>> +
>> +    buf[nbytes] = 0;
>> +    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;
>> +}
>>
>

```

>  
> These bits look a little out of sync, with no users for these routines in  
> this patch. Won't you get a compiler warning, compiling this bit alone?  
>

Nope - when you have a non-static function without users in a  
file no compiler warning produced.

---