

---

Subject: Re: [PATCH 1/9] Containers (V9): Basic container framework

Posted by [Balbir Singh](#) on Tue, 01 May 2007 17:40:52 GMT

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

---

menage@google.com wrote:

> This patch adds the main containers framework - the container  
> filesystem, and the basic structures for tracking membership and  
> associating subsystem state objects to tasks.

[snip]

> +\*\*\* notify\_on\_release is disabled in the current patch set. It may be  
> +\*\*\* reactivated in a future patch in a less-intrusive manner  
> +

Won't this break user space tools for cpusets?

[snip]

> +See kernel/container.c for more details.  
> +  
> +Subsystems can take/release the container\_mutex via the functions  
> +container\_lock()/container\_unlock(), and can  
> +take/release the callback\_mutex via the functions  
> +container\_lock()/container\_unlock().  
> +

Hmm.. looks like a documentation error. Both mutex's are obtained through  
container\_lock/container\_unlock ?

> +Accessing a task's container pointer may be done in the following ways:  
> +- while holding container\_mutex  
> +- while holding the task's alloc\_lock (via task\_lock())  
> +- inside an rcu\_read\_lock() section via rcu\_dereference()  
> +

container\_mutex() and task\_lock() can be used for changing the pointer?  
Could you please explain this a bit further.

[snip]

> +int populate(struct container\_subsys \*ss, struct container \*cont)  
> +LL=none  
> +  
> +Called after creation of a container to allow a subsystem to populate  
> +the container directory with file entries. The subsystem should make  
> +calls to container\_add\_file() with objects of type cftype (see  
> +include/linux/container.h for details). Note that although this

> +method can return an error code, the error code is currently not  
> +always handled well.

We needed the equivalent of container\_remove\_file() to be called  
if container\_add\_file() failed.

[snip]

```
> +struct container {
> + unsigned long flags; /* "unsigned long" so bitops work */
> +
> + /* count users of this container. >0 means busy, but doesn't
> +  * necessarily indicate the number of tasks in the
> +  * container */
> + atomic_t count;
> +
> + /*
> +  * We link our 'sibling' struct into our parent's 'children'.
> +  * Our children link their 'sibling' into our 'children'.
> +  */
> + struct list_head sibling; /* my parent's children */
> + struct list_head children; /* my children */
> +
> + struct container *parent; /* my parent */
> + struct dentry *dentry; /* container fs entry */
> +
> + /* Private pointers for each registered subsystem */
> + struct container_subsys_state *subsys[CONTAINER_SUBSYS_COUNT];
> +
> + struct containerfs_root *root;
> + struct container *top_container;
> +};
```

Can't we derive the top\_container from containerfs\_root?

```
> +
> +/* struct cftype:
> + *
> + * The files in the container filesystem mostly have a very simple read/write
> + * handling, some common function will take care of it. Nevertheless some cases
> + * (read tasks) are special and therefore I define this structure for every
> + * kind of file.
> + *
> + *
> + * When reading/writing to a file:
> + * - the container to use in file->f_dentry->d_parent->d_fsdata
> + * - the 'cftype' of the file is file->f_dentry->d_fsdata
```

```
> + */
> +
> +struct inode;
> +#define MAX_CFTYPE_NAME 64
> +struct cftype {
> + /* By convention, the name should begin with the name of the
> +  * subsystem, followed by a period */
> + char name[MAX_CFTYPE_NAME];
> + int private;
> + int (*open) (struct inode *inode, struct file *file);
> + ssize_t (*read) (struct container *cont, struct cftype *cft,
> +   struct file *file,
> +   char __user *buf, size_t nbytes, loff_t *ppos);
> + u64 (*read_uint) (struct container *cont, struct cftype *cft);
```

Is this a new callback, a specialization of the read() callback?

```
> + ssize_t (*write) (struct container *cont, struct cftype *cft,
> +   struct file *file,
> +   const char __user *buf, size_t nbytes, loff_t *ppos);
> + int (*release) (struct inode *inode, struct file *file);
> +};
> +
```

[snip]

--

Warm Regards,  
Balbir Singh  
Linux Technology Center  
IBM, ISTL

---