Subject: [PATCH][DOCUMENTATION] Minimal controller code for a quick start
Posted by Pavel Emelianov on Thu, 07 Feb 2008 15:37:17 GMT
View Forum Message <> Reply to Message

The Documentation/cgroups.txt file contains the info on how
to write some controller for cgroups subsystem, but even with
this, one need to write quite a lot of code before developing
the core (or copy-n-paste it from some other place).

I propose to put this minimal controller into Documentation
directory to let people copy-n-paste a) from a known place and
b) a small piece of code.

Besides, many people learn better reading an example rather
than/along with a document.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/Documentation/cgroups.txt b/Documentation/cgroups.txt
index 42d7c4c..66068dc 100644
--- a/Documentation/cgroups.txt
+++ b/Documentation/cgroups.txt
@@ -531,6 +531,9 @@ and root cgroup. Currently this will only involve movement between
 the default hierarchy (which never has sub-cgroups) and a hierarchy
 that is being created/destroyed (and hence has no sub-cgroups).

+For a quick start you may want to look at the
+Documentation/controllers/example.c file.
+
 4. Questions
 ============

diff --git a/Documentation/controllers/example.c b/Documentation/controllers/example.c
new file mode 100644
index 0000000..4a73c77
--- /dev/null
+++ b/Documentation/controllers/example.c
@@ -0,0 +1,134 @@
+/*
+ * Documentation/controllers/example.c - A simple controller
+ *
+ * Copy and make s/foo/$SUBSYS_NAME/g in it to get a minimal
+ * working code. Don't forget to add a SUBSYS(foo) line in the
+ * include/linux/cgroup_subsys.h file.
+ *
+ */
```

```
+
+#include <linux/cgroup.h>
+
+/*
+ * the foo main structure - it is used to store any info, that
+ * is required from the group of tasks
+ */
+
+struct foo_cgroup {
+ /*
+  * put your fields here
+  */
+
+ struct cgroup_subsys_state css;
+
+ /*
+  * ... or/and here
+  */
+};
+
+/*
+ * helpers to get the foo_cgroup from a task and a control group
+ */
+
+static inline struct foo_cgroup *foo_from_css(struct cgroup_subsys_state *css)
+{
+ return container_of(css, struct foo_cgroup, css);
+}
+
+static inline struct foo_cgroup *foo_from_cgroup(struct cgroup *cg)
+{
+ return foo_from_css(cgroup_subsys_state(cg, foo_subsys_id));
+}
+
+static inline struct foo_cgroup *foo_from_task(struct task_struct *p)
+{
+ return foo_from_css(task_subsys_state(p, foo_subsys_id));
+}
+
+/*
+ * foo files
+ */
+
+static ssize_t foo_bar_read(struct cgroup *cg, struct cftype *cft,
+  struct file *file, char __user *userbuf,
+  size_t nbytes, loff_t *ppos)
+{
+ struct foo_cgroup *foo;
```

```
+
+ foo = foo_from_cgroup(cg);
+
+ /*
+  * produce some output
+  */
+
+ return nbytes;
+}
+
+static ssize_t foo_bar_write(struct cgroup *cg, struct cftype *cft,
+  struct file *file, const char __user *userbuf,
+  size_t nbytes, loff_t *ppos)
+{
+ struct foo_cgroup *foo;
+
+ foo = foo_from_cgroup(cg);
+
+ /*
+  * read and tune the foo
+  */
+
+ return nbytes;
+}
+
+static struct cftype foo_files[] = {
+ {
+  .name = "bar",
+  .read = foo_bar_read,
+  .write = foo_bar_write,
+ },
+};
+
+/*
+ * foo subsystem basic callbacks
+ */
+
+static struct cgroup_subsys_state *foo_create(struct cgroup_subsys *cs,
+  struct cgroup *cg)
+{
+ struct foo_cgroup *foo;
+
+ foo = kmalloc(sizeof(struct foo_cgroup), GFP_KERNEL);
+ if (foo == NULL)
+  return NULL;
+
+ /*
+  * initialize your fields
```

```
+  */
+
+ return &foo->css;
+}
+
+static void foo_destroy(struct cgroup_subsys *cs, struct cgroup *cg)
+{
+ struct foo_cgroup *foo;
+
+ foo = foo_from_cgroup(cg);
+
+ /*
+  * clean your fields
+  */
+
+ kfree(foo);
+}
+
+static int foo_populate(struct cgroup_subsys *cs, struct cgroup *cg)
+{
+ return cgroup_add_files(cg, cs, foo_files, ARRAY_SIZE(foo_files));
+}
+
+struct cgroup_subsys foo_subsys = {
+ .name = "foo",
+ .subsys_id = foo_subsys_id,
+ .create = foo_create,
+ .destroy = foo_destroy,
+ .populate = foo_populate,
+};
```

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

---

Subject: Re: [PATCH][DOCUMENTATION] Minimal controller code for a quick start
Posted by Peter Zijlstra on Thu, 07 Feb 2008 20:28:57 GMT
View Forum Message <> Reply to Message

On Thu, 2008-02-07 at 18:37 +0300, Pavel Emelyanov wrote:
> The Documentation/cgroups.txt file contains the info on how
> to write some controller for cgroups subsystem, but even with
> this, one need to write quite a lot of code before developing
> the core (or copy-n-paste it from some other place).
>
> I propose to put this minimal controller into Documentation
> directory to let people copy-n-paste a) from a known place and

> b) a small piece of code.
>
> Besides, many people learn better reading an example rather
> than/along with a document.

While on the subject, could someone document struct cgroup_subsys. In particular, I've wondered why we have: cgroup_subsys::can_attach() and not use a return value in cgroup_subsys::attach()?

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

---

Subject: Re: [PATCH][DOCUMENTATION] Minimal controller code for a quick start
Posted by Paul Menage on Thu, 07 Feb 2008 20:45:18 GMT
View Forum Message <> Reply to Message

On Feb 7, 2008 7:37 AM, Pavel Emelyanov <xemul@openvz.org> wrote:
> The Documentation/cgroups.txt file contains the info on how
> to write some controller for cgroups subsystem, but even with
> this, one need to write quite a lot of code before developing
> the core (or copy-n-paste it from some other place).

Good idea.

```
> +
> +static ssize_t foo_bar_read(struct cgroup *cg, struct cftype *cft,
> +                struct file *file, char __user *userbuf,
> +                size_t nbytes, loff_t *ppos)
> +{
> +       struct foo_cgroup *foo;
> +
> +       foo = foo_from_cgroup(cg);
> +
> +       /*
> +        * produce some output
> +        */
> +
> +       return nbytes;
> +}
> +
> +static ssize_t foo_bar_write(struct cgroup *cg, struct cftype *cft,
> +                struct file *file, const char __user *userbuf,
```

```
> +                size_t nbytes, loff_t *ppos)
> +{
> +       struct foo_cgroup *foo;
> +
> +       foo = foo_from_cgroup(cg);
> +
> +       /*
> +        * read and tune the foo
> +        */
> +
> +       return nbytes;
> +}
> +
> +static struct cftype foo_files[] = {
> +       {
> +               .name = "bar",
> +               .read = foo_bar_read,
> +               .write = foo_bar_write,
> +       },
> +};
```

Can you structure this example so as to encourage people to use the more formatted read/write routines, such as read_int64 and write_int64?

```
> +
> +static struct cgroup_subsys_state *foo_create(struct cgroup_subsys *cs,
> +               struct cgroup *cg)
> +{
> +       struct foo_cgroup *foo;
> +
```

Maybe add a comment here that mentions that if your cgroup needs very early initialization, you can check for cg->parent being NULL, and return a statically-constructed structure here. (And set foo_subsys.early_init = 1)

Paul

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

---

Subject: Re: [PATCH][DOCUMENTATION] Minimal controller code for a quick start
Posted by serue on Thu, 07 Feb 2008 20:47:04 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> The Documentation/cgroups.txt file contains the info on how
> to write some controller for cgroups subsystem, but even with
> this, one need to write quite a lot of code before developing
> the core (or copy-n-paste it from some other place).
>
> I propose to put this minimal controller into Documentation
> directory to let people copy-n-paste a) from a known place and
> b) a small piece of code.
>
> Besides, many people learn better reading an example rather
> than/along with a document.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Actually I thought that was the main point of kernel/cgroup_debug.c?

-serge

>
> ---
>
> diff --git a/Documentation/cgroups.txt b/Documentation/cgroups.txt
> index 42d7c4c..66068dc 100644
> --- a/Documentation/cgroups.txt
> +++ b/Documentation/cgroups.txt
> @@ -531,6 +531,9 @@ and root cgroup. Currently this will only involve movement between
> the default hierarchy (which never has sub-cgroups) and a hierarchy
> that is being created/destroyed (and hence has no sub-cgroups).
>
> +For a quick start you may want to look at the
> +Documentation/controllers/example.c file.
> +
> 4. Questions
> ============
>
> diff --git a/Documentation/controllers/example.c b/Documentation/controllers/example.c
> new file mode 100644
> index 0000000..4a73c77
> --- /dev/null
> +++ b/Documentation/controllers/example.c
> @@ -0,0 +1,134 @@
> +/*
> + * Documentation/controllers/example.c - A simple controller
> + *
> + * Copy and make s/foo/$SUBSYS_NAME/g in it to get a minimal
> + * working code. Don't forget to add a SUBSYS(foo) line in the
> + * include/linux/cgroup_subsys.h file.

```
> + *
> + */
> +
> +#include <linux/cgroup.h>
> +
> +/*
> + * the foo main structure - it is used to store any info, that
> + * is required from the group of tasks
> + */
> +
> +struct foo_cgroup {
> + /*
> +  * put your fields here
> +  */
> +
> + struct cgroup_subsys_state css;
> +
> + /*
> +  * ... or/and here
> +  */
> +};
> +
> +/*
> + * helpers to get the foo_cgroup from a task and a control group
> + */
> +
> +static inline struct foo_cgroup *foo_from_css(struct cgroup_subsys_state *css)
> +{
> + return container_of(css, struct foo_cgroup, css);
> +}
> +
> +static inline struct foo_cgroup *foo_from_cgroup(struct cgroup *cg)
> +{
> + return foo_from_css(cgroup_subsys_state(cg, foo_subsys_id));
> +}
> +
> +static inline struct foo_cgroup *foo_from_task(struct task_struct *p)
> +{
> + return foo_from_css(task_subsys_state(p, foo_subsys_id));
> +}
> +
> +/*
> + * foo files
> + */
> +
> +static ssize_t foo_bar_read(struct cgroup *cg, struct cftype *cft,
> +  struct file *file, char __user *userbuf,
> +  size_t nbytes, loff_t *ppos)
```

```
> +{
> + struct foo_cgroup *foo;
> +
> + foo = foo_from_cgroup(cg);
> +
> + /*
> +  * produce some output
> +  */
> +
> + return nbytes;
> +}
> +
> +static ssize_t foo_bar_write(struct cgroup *cg, struct cftype *cft,
> +  struct file *file, const char __user *userbuf,
> +  size_t nbytes, loff_t *ppos)
> +{
> + struct foo_cgroup *foo;
> +
> + foo = foo_from_cgroup(cg);
> +
> + /*
> +  * read and tune the foo
> +  */
> +
> + return nbytes;
> +}
> +
> +static struct cftype foo_files[] = {
> + {
> +  .name = "bar",
> +  .read = foo_bar_read,
> +  .write = foo_bar_write,
> + },
> +};
> +
> +/*
> + * foo subsystem basic callbacks
> + */
> +
> +static struct cgroup_subsys_state *foo_create(struct cgroup_subsys *cs,
> +  struct cgroup *cg)
> +{
> + struct foo_cgroup *foo;
> +
> + foo = kmalloc(sizeof(struct foo_cgroup), GFP_KERNEL);
> + if (foo == NULL)
> +  return NULL;
> +
```

```
> + /*
> +  * initialize your fields
> +  */
> +
> + return &foo->css;
> +}
> +
> +static void foo_destroy(struct cgroup_subsys *cs, struct cgroup *cg)
> +{
> + struct foo_cgroup *foo;
> +
> + foo = foo_from_cgroup(cg);
> +
> + /*
> +  * clean your fields
> +  */
> +
> + kfree(foo);
> +}
> +
> +static int foo_populate(struct cgroup_subsys *cs, struct cgroup *cg)
> +{
> + return cgroup_add_files(cg, cs, foo_files, ARRAY_SIZE(foo_files));
> +}
> +
> +struct cgroup_subsys foo_subsys = {
> + .name = "foo",
> + .subsys_id = foo_subsys_id,
> + .create = foo_create,
> + .destroy = foo_destroy,
> + .populate = foo_populate,
> +};
> _____________________________________________________
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers

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

Subject: Re: [PATCH][DOCUMENTATION] Minimal controller code for a quick start
Posted by Paul Menage on Thu, 07 Feb 2008 20:49:25 GMT

View Forum Message <> Reply to Message

On Feb 7, 2008 12:28 PM, Peter Zijlstra <a.p.zijlstra@chello.nl> wrote:
>

> While on the subject, could someone document struct cgroup_subsys.

There's documentation for all the methods in Documentation/cgroup.txt

> particular, I've wondered why we have: cgroup_subsys::can_attach() and
> not use a return value in cgroup_subsys::attach()?

We could do in theory do that, but it would make the recovery logic in
cgroup.c:attach_task() more complex - it would have to be able to deal
with undoing a partial attach. It seems simpler to just split it into
two phases, given that most cgroups don't appear to have attachment
conditions anyway.

Paul

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

---

Subject: Re: [PATCH][DOCUMENTATION] Minimal controller code for a quick start
Posted by serue on Thu, 07 Feb 2008 20:50:21 GMT
View Forum Message <> Reply to Message

Quoting Peter Zijlstra (a.p.zijlstra@chello.nl):
>
> On Thu, 2008-02-07 at 18:37 +0300, Pavel Emelyanov wrote:
> > > The Documentation/cgroups.txt file contains the info on how
> > > to write some controller for cgroups subsystem, but even with
> > > this, one need to write quite a lot of code before developing
> > > the core (or copy-n-paste it from some other place).
> > >
> > > I propose to put this minimal controller into Documentation
> > > directory to let people copy-n-paste a) from a known place and
> > > b) a small piece of code.
> > >
> > > Besides, many people learn better reading an example rather
> > > than/along with a document.
> >
> >
> > While on the subject, could someone document struct cgroup_subsys. In
> > particular, I've wondered why we have: cgroup_subsys::can_attach() and

The point of can_attach() is to decide whether an attach should be
permitted. kernel/ns_cgroup.c defines it.

> not use a return value in cgroup_subsys::attach()?

IIUC, by the point ->attach() is called, the task has already been placed
in the new cgroup, and we're just asking each subsystem to update any
relevant accounting.

-serge

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

---

Subject: Re: [PATCH][DOCUMENTATION] Minimal controller code for a quick start
Posted by Peter Zijlstra on Thu, 07 Feb 2008 20:52:33 GMT
View Forum Message <> Reply to Message

On Thu, 2008-02-07 at 12:49 -0800, Paul Menage wrote:
> On Feb 7, 2008 12:28 PM, Peter Zijlstra <a.p.zijlstra@chello.nl> wrote:
> >
> > While on the subject, could someone document struct cgroup_subsys.
>
> There's documentation for all the methods in Documentation/cgroup.txt

Hehe, and here I was looking for in-code comments. OK I'll read the
thing.

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

---

Subject: Re: [PATCH][DOCUMENTATION] Minimal controller code for a quick start
Posted by Pavel Emelianov on Fri, 08 Feb 2008 08:09:09 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On Feb 7, 2008 7:37 AM, Pavel Emelyanov <xemul@openvz.org> wrote:
>> The Documentation/cgroups.txt file contains the info on how
>> to write some controller for cgroups subsystem, but even with
>> this, one need to write quite a lot of code before developing
>> the core (or copy-n-paste it from some other place).
>
> Good idea.
>
>> +
>> +static ssize_t foo_bar_read(struct cgroup *cg, struct cftype *cft,
>> +   struct file *file, char __user *userbuf,

```
>> +               size_t nbytes, loff_t *ppos)
>> +{
>> +       struct foo_cgroup *foo;
>> +
>> +       foo = foo_from_cgroup(cg);
>> +
>> +       /*
>> +        * produce some output
>> +        */
>> +
>> +       return nbytes;
>> +}
>> +
>> +static ssize_t foo_bar_write(struct cgroup *cg, struct cftype *cft,
>> +               struct file *file, const char __user *userbuf,
>> +               size_t nbytes, loff_t *ppos)
>> +{
>> +       struct foo_cgroup *foo;
>> +
>> +       foo = foo_from_cgroup(cg);
>> +
>> +       /*
>> +        * read and tune the foo
>> +        */
>> +
>> +       return nbytes;
>> +}
>> +
>> +static struct cftype foo_files[] = {
>> +       {
>> +               .name = "bar",
>> +               .read = foo_bar_read,
>> +               .write = foo_bar_write,
>> +       },
>> +};
>
> Can you structure this example so as to encourage people to use the
> more formatted read/write routines, such as read_int64 and
> write_int64?
```

And one more for ->read_strategy hints :) Well, I thought that
it should be the _minimal_ controller.

```
>> +
>> +static struct cgroup_subsys_state *foo_create(struct cgroup_subsys *cs,
>> +               struct cgroup *cg)
>> +{
>> +       struct foo_cgroup *foo;
```

>> +
>
> Maybe add a comment here that mentions that if your cgroup needs very
> early initialization, you can check for cg->parent being NULL, and
> return a statically-constructed structure here. (And set
> foo_subsys.early_init = 1)

Yet again - this is rather a special feature, that your controller
needs an early initialization - for a very minimal one this is not
required.

Maybe we can have two examples? One is the minimal one and the other
is an advanced one with ->attach callbacks, etc?

> Paul
>

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

---

Subject: Re: [PATCH][DOCUMENTATION] Minimal controller code for a quick start
Posted by Pavel Emelianov on Fri, 08 Feb 2008 09:28:26 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelyanov (xemul@openvz.org):
>> The Documentation/cgroups.txt file contains the info on how
>> to write some controller for cgroups subsystem, but even with
>> this, one need to write quite a lot of code before developing
>> the core (or copy-n-paste it from some other place).
>>
>> I propose to put this minimal controller into Documentation
>> directory to let people copy-n-paste a) from a known place and
>> b) a small piece of code.
>>
>> Besides, many people learn better reading an example rather
>> than/along with a document.
>>
>> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> Actually I thought that was the main point of kernel/cgroup_debug.c?

This one doesn't show how to use generic read/write for files,
but a bit more advanced read_uint/write_uiant, it doesn't declare
its own cgroup with extra fields and doesn't show how to get the

one from task.

Hmm... This one is even more minimal than my :)

> -serge
>
>> ---
>>
>> diff --git a/Documentation/cgroups.txt b/Documentation/cgroups.txt
>> index 42d7c4c..66068dc 100644
>> --- a/Documentation/cgroups.txt
>> +++ b/Documentation/cgroups.txt
>> @@ -531,6 +531,9 @@ and root cgroup. Currently this will only involve movement between
>> the default hierarchy (which never has sub-cgroups) and a hierarchy
>> that is being created/destroyed (and hence has no sub-cgroups).
>>
>> +For a quick start you may want to look at the
>> +Documentation/controllers/example.c file.
>> +
>> 4. Questions
>> ============
>>
>> diff --git a/Documentation/controllers/example.c b/Documentation/controllers/example.c
>> new file mode 100644
>> index 0000000..4a73c77
>> --- /dev/null
>> +++ b/Documentation/controllers/example.c
>> @@ -0,0 +1,134 @@
>> +/*
>> + * Documentation/controllers/example.c - A simple controller
>> + *
>> + * Copy and make s/foo/$SUBSYS_NAME/g in it to get a minimal
>> + * working code. Don't forget to add a SUBSYS(foo) line in the
>> + * include/linux/cgroup_subsys.h file.
>> + *
>> + */
>> +
>> +#include <linux/cgroup.h>
>> +
>> +/*
>> + * the foo main structure - it is used to store any info, that
>> + * is required from the group of tasks
>> + */
>> +
>> +struct foo_cgroup {
>> + /*
>> + * put your fields here
>> + */

```
>> +
>> + struct cgroup_subsys_state css;
>> +
>> + /*
>> +  * ... or/and here
>> +  */
>> +};
>> +
>> +/*
>> + * helpers to get the foo_cgroup from a task and a control group
>> + */
>> +
>> +static inline struct foo_cgroup *foo_from_css(struct cgroup_subsys_state *css)
>> +{
>> + return container_of(css, struct foo_cgroup, css);
>> +}
>> +
>> +static inline struct foo_cgroup *foo_from_cgroup(struct cgroup *cg)
>> +{
>> + return foo_from_css(cgroup_subsys_state(cg, foo_subsys_id));
>> +}
>> +
>> +static inline struct foo_cgroup *foo_from_task(struct task_struct *p)
>> +{
>> + return foo_from_css(task_subsys_state(p, foo_subsys_id));
>> +}
>> +
>> +/*
>> + * foo files
>> + */
>> +
>> +static ssize_t foo_bar_read(struct cgroup *cg, struct cftype *cft,
>> +  struct file *file, char __user *userbuf,
>> +  size_t nbytes, loff_t *ppos)
>> +{
>> + struct foo_cgroup *foo;
>> +
>> + foo = foo_from_cgroup(cg);
>> +
>> + /*
>> +  * produce some output
>> +  */
>> +
>> + return nbytes;
>> +}
>> +
>> +static ssize_t foo_bar_write(struct cgroup *cg, struct cftype *cft,
>> +  struct file *file, const char __user *userbuf,
```

```
>> +  size_t nbytes, loff_t *ppos)
>> +{
>> + struct foo_cgroup *foo;
>> +
>> + foo = foo_from_cgroup(cg);
>> +
>> + /*
>> +  * read and tune the foo
>> +  */
>> +
>> + return nbytes;
>> +}
>> +
>> +static struct cftype foo_files[] = {
>> + {
>> +  .name = "bar",
>> +  .read = foo_bar_read,
>> +  .write = foo_bar_write,
>> + },
>> +};
>> +
>> +/*
>> + * foo subsystem basic callbacks
>> + */
>> +
>> +static struct cgroup_subsys_state *foo_create(struct cgroup_subsys *cs,
>> +  struct cgroup *cg)
>> +{
>> + struct foo_cgroup *foo;
>> +
>> + foo = kmalloc(sizeof(struct foo_cgroup), GFP_KERNEL);
>> + if (foo == NULL)
>> +  return NULL;
>> +
>> + /*
>> +  * initialize your fields
>> +  */
>> +
>> + return &foo->css;
>> +}
>> +
>> +static void foo_destroy(struct cgroup_subsys *cs, struct cgroup *cg)
>> +{
>> + struct foo_cgroup *foo;
>> +
>> + foo = foo_from_cgroup(cg);
>> +
>> + /*
```

>> + * clean your fields
>> + */
>> +
>> + kfree(foo);
>> +}
>> +
>> +static int foo_populate(struct cgroup_subsys *cs, struct cgroup *cg)
>> +{
>> + return cgroup_add_files(cg, cs, foo_files, ARRAY_SIZE(foo_files));
>> +}
>> +
>> +struct cgroup_subsys foo_subsys = {
>> + .name = "foo",
>> + .subsys_id = foo_subsys_id,
>> + .create = foo_create,
>> + .destroy = foo_destroy,
>> + .populate = foo_populate,
>> +};
>> _____
>> Containers mailing list
>> Containers@lists.linux-foundation.org
>> https://lists.linux-foundation.org/mailman/listinfo/containers
>

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