Subject: Re: [PATCH 4/4] The control group itself
Posted by Paul Menage on Mon, 14 Jan 2008 21:54:33 GMT
View Forum Message <> Reply to Message

On Jan 8, 2008 1:18 AM, Pavel Emelyanov <xemul@openvz.org> wrote:
> Each new group will have its own maps for char and block
> layers. The devices access list is tuned via the
> devices.permissions file.
>
> One may read from the file to get the configured
> state. E.g.
>
> # cat /cont/devices/0/devices.permissions
> c 1:* rw
> b 8:1 rw
>
> The top container isn't initialized, so that the char
> and block layers will use the global maps to lookup
> their devices.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> ---
>
> diff --git a/fs/Makefile b/fs/Makefile
> index 82b6ae1..a085706 100644
> --- a/fs/Makefile
> +++ b/fs/Makefile
> @@ -63,6 +63,8 @@ obj-y   += devpts/
>
> obj-$(CONFIG_PROFILING)   += dcookies.o
> obj-$(CONFIG_DLM)   += dlm/
> +
> +obj-$(CONFIG_CGROUP_DEVS)   += devscontrol.o
>
> # Do not add any filesystems before this line
> obj-$(CONFIG_REISERFS_FS)   += reiserfs/
> diff --git a/fs/devscontrol.c b/fs/devscontrol.c
> new file mode 100644
> index 0000000..ea282f3
> --- /dev/null
> +++ b/fs/devscontrol.c
> @@ -0,0 +1,291 @@
> +/*
> + * devscontrol.c - Device Controller
> + *
> + * Copyright 2007 OpenVZ SWsoft Inc
> + * Author: Pavel Emelyanov <xemul at openvz.org>

```
> + *
> + * This program is free software; you can redistribute it and/or modify
> + * it under the terms of the GNU General Public License as published by
> + * the Free Software Foundation; either version 2 of the License, or
> + * (at your option) any later version.
> + *
> + * This program is distributed in the hope that it will be useful,
> + * but WITHOUT ANY WARRANTY; without even the implied warranty of
> + * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
> + * GNU General Public License for more details.
> + */
> +
> +#include <linux/cgroup.h>
> +#include <linux/cdev.h>
> +#include <linux/err.h>
> +#include <linux/devscontrol.h>
> +#include <linux/uaccess.h>
> +#include <linux/fs.h>
> +#include <linux/genhd.h>
> +
> +struct devs_cgroup {
> +       struct cgroup_subsys_state css;
> +
> +       struct kobj_map *cdev_map;
> +       struct kobj_map *bdev_map;
> +};
> +
> +static inline
> +struct devs_cgroup *css_to_devs(struct cgroup_subsys_state *css)
> +{
> +       return container_of(css, struct devs_cgroup, css);
> +}
> +
> +static inline
> +struct devs_cgroup *cgroup_to_devs(struct cgroup *cont)
> +{
> +       return css_to_devs(cgroup_subsys_state(cont, devs_subsys_id));
> +}
> +
> +struct kobj_map *task_cdev_map(struct task_struct *tsk)
> +{
> +       struct cgroup_subsys_state *css;
> +
> +       css = task_subsys_state(tsk, devs_subsys_id);
> +       if (css->cgroup->parent == NULL)
> +               return NULL;
> +       else
> +               return css_to_devs(css)->cdev_map;
```

> +}

For this (and task_bdev_map) it would be cleaner to just leave the
cdev_map and bdev_map in the root cgroup as NULL, so you could just
make this

```
struct kobj_map *task_cdev_map(struct task_struct *tsk)
{
  return css_to_devs(task_subsys_state(tsk, devs_subsys_id))->cdev_map;
}
> +
> +static ssize_t devs_write(struct cgroup *cont, struct cftype *cft,
> +             struct file *f, const char __user *ubuf,
> +             size_t nbytes, loff_t *pos)
> +{
> +       int err, all, chrdev;
> +       dev_t dev;
> +       char buf[64];
> +       struct devs_cgroup *devs;
> +       mode_t mode;
> +
> +       if (copy_from_user(buf, ubuf, sizeof(buf)))
> +               return -EFAULT;
> +
> +       buf[sizeof(buf) - 1] = 0;
> +       err = decode_perms_str(buf, &chrdev, &dev, &all, &mode);
> +       if (err < 0)
> +               return err;
> +
> +       devs = cgroup_to_devs(cont);
> +
> +       if (mode == 0) {
> +               if (chrdev)
> +                       err = cdev_del_from_map(devs->cdev_map, dev, all);
> +               else
> +                       err = bdev_del_from_map(devs->bdev_map, dev, all);
```

There's no locking involved on these calls, other than the cgroups
code guaranteeing that the subsystem objects themselves won't go away.
A quick look over the kobj_map calls suggests that these calls may be
thread-safe - can you confirm that. (And maybe comment at the top of
the function?)

```
> +       arg.buf = (char *)__get_free_page(GFP_KERNEL);
> +       if (arg.buf == NULL)
> +               return -ENOMEM;
> +
> +       devs = cgroup_to_devs(cont);
```

```
> +       arg.pos = 0;
> +
> +       arg.chrdev = 1;
> +       cdev_iterate_map(devs->cdev_map, devs_dump_one, &arg);
> +
> +       arg.chrdev = 0;
> +       bdev_iterate_map(devs->bdev_map, devs_dump_one, &arg);
```

Is there any chance of this overflowing the buffer page?

Paul