Subject: Re: [PATCH 4/4] The control group itself
Posted by serue on Mon, 14 Jan 2008 17:40:56 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> 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;
```

```
> +}
> +
> +struct kobj_map *task_bdev_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)->bdev_map;
> +}
> +
> +static struct cgroup_subsys_state *
> +devs_create(struct cgroup_subsys *ss, struct cgroup *cont)
> +{
> + struct devs_cgroup *devs;
> +
> + devs = kzalloc(sizeof(struct devs_cgroup), GFP_KERNEL);
> + if (devs == NULL)
> +  goto out;
> +
> + devs->cdev_map = cdev_map_init();
> + if (devs->cdev_map == NULL)
> +  goto out_free;
> +
> + devs->bdev_map = bdev_map_init();
> + if (devs->bdev_map == NULL)
> +  goto out_free_cdev;
> +
> + return &devs->css;
> +
> +out_free_cdev:
> + cdev_map_fini(devs->cdev_map);
> +out_free:
> + kfree(devs);
> +out:
> + return ERR_PTR(-ENOMEM);
> +}
```

Thanks for working on this, Pavel.

My only question with this patch is - so if I create a devs cgroup which only has access to, say /dev/loop0 and /dev/tty3, and someone in that cgroup manages to create a new cgroup, the new cgroup will have all the default permissions again, rather than inherit the permissions from this cgroup, right?

```
> +
> +static void devs_destroy(struct cgroup_subsys *ss, struct cgroup *cont)
> +{
> + struct devs_cgroup *devs;
> +
> + devs = cgroup_to_devs(cont);
> + bdev_map_fini(devs->bdev_map);
> + cdev_map_fini(devs->cdev_map);
> + kfree(devs);
> +}
> +
> +static int decode_perms_str(char *buf, int *chrdev, dev_t *dev,
> +  int *all, mode_t *mode)
> +{
> + unsigned int major, minor;
> + char *end;
> + mode_t tmp;
> +
> + /* [cb] <major>:<minor> [r-][w-] */
> +
> + if (buf[0] == 'c')
> +  *chrdev = 1;
> + else if (buf[0] == 'b')
> +  *chrdev = 0;
> + else
> +  return -EINVAL;
> +
> + if (buf[1] != ' ')
> +  return -EINVAL;
> +
> + major = simple_strtoul(buf + 2, &end, 10);
> + if (*end != ':')
> +  return -EINVAL;
> +
> + if (end[1] == '*') {
> +  if (end[2] != ' ')
> +   return -EINVAL;
> +
> +  *all = 1;
> +  minor = 0;
> +  end += 2;
> + } else {
> +  minor = simple_strtoul(end + 1, &end, 10);
> +  if (*end != ' ')
> +   return -EINVAL;
> +
> +  *all = 0;
> + }
```

```
> +
> + tmp = 0;
> +
> + if (end[1] == 'r')
> +  tmp |= FMODE_READ;
> + else if (end[1] != '-')
> +  return -EINVAL;
> + if (end[2] == 'w')
> +  tmp |= FMODE_WRITE;
> + else if (end[2] != '-')
> +  return -EINVAL;
> +
> + *dev = MKDEV(major, minor);
> + *mode = tmp;
> + return 0;
> +}
> +
> +static int encode_perms_str(char *buf, int len, int chrdev, dev_t dev,
> +  int all, mode_t mode)
> +{
> + int ret;
> +
> + ret = snprintf(buf, len, "%c %d:", chrdev ? 'c' : 'b', MAJOR(dev));
> + if (all)
> +  ret += snprintf(buf + ret, len - ret, "*");
> + else
> +  ret += snprintf(buf + ret, len - ret, "%d", MINOR(dev));
> +
> + ret += snprintf(buf + ret, len - ret, " %c%c\n",
> +   (mode & FMODE_READ) ? 'r' : '-',
> +   (mode & FMODE_WRITE) ? 'w' : '-');
> +
> + return ret + 1;
> +}
> +
> +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;
```

(Of course this will require some privilege, i assume that's a detail
you'll add next time around)

```
> +
> + 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);
> +
> +  if (err < 0)
> +   return err;
> +
> +  css_put(&devs->css);
> + } else {
> +  if (chrdev)
> +   err = cdev_add_to_map(devs->cdev_map, dev, all, mode);
> +  else
> +   err = bdev_add_to_map(devs->bdev_map, dev, all, mode);
> +
> +  if (err < 0)
> +   return err;
> +
> +  css_get(&devs->css);
> + }
> +
> + return nbytes;
> +}
> +
> +struct devs_dump_arg {
> + char *buf;
> + int pos;
> + int chrdev;
> +};
> +
> +static int devs_dump_one(dev_t dev, int range, mode_t mode, void *x)
> +{
> + struct devs_dump_arg *arg = x;
> + char tmp[64];
> + int len;
> +
```

```
> + len = encode_perms_str(tmp, sizeof(tmp), arg->chrdev, dev,
> +   range != 1, mode);
> +
> + if (arg->pos >= PAGE_SIZE - len)
> +  return 1;
> +
> + memcpy(arg->buf + arg->pos, tmp, len);
> + arg->pos += len;
> + return 0;
> +}
> +
> +static ssize_t devs_read(struct cgroup *cont, struct cftype *cft,
> +  struct file *f, char __user *ubuf, size_t nbytes, loff_t *pos)
> +{
> + struct devs_dump_arg arg;
> + struct devs_cgroup *devs;
> + ssize_t ret;
> +
> + 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);
> +
> + ret = simple_read_from_buffer(ubuf, nbytes, pos,
> +   arg.buf, arg.pos);
> +
> + free_page((unsigned long)arg.buf);
> + return ret;
> +}
> +
> +static struct cftype devs_files[] = {
> + {
> +  .name = "permissions",
> +  .write = devs_write,
> +  .read = devs_read,
> + },
> +};
> +
> +static int devs_populate(struct cgroup_subsys *ss, struct cgroup *cont)
> +{
```

> + return cgroup_add_files(cont, ss,
> +   devs_files, ARRAY_SIZE(devs_files));
> +}
> +
> +struct cgroup_subsys devs_subsys = {
> + .name = "devices",
> + .subsys_id = devs_subsys_id,
> + .create = devs_create,
> + .destroy = devs_destroy,
> + .populate = devs_populate,
> +};
> diff --git a/include/linux/cgroup_subsys.h b/include/linux/cgroup_subsys.h
> index 228235c..9c0cd2c 100644
> --- a/include/linux/cgroup_subsys.h
> +++ b/include/linux/cgroup_subsys.h
> @@ -42,3 +42,9 @@ SUBSYS(mem_cgroup)
> #endif
>
> /* */
> +
> +#ifdef CONFIG_CGROUP_DEVS
> +SUBSYS(devs)
> +#endif
> +
> +/* */
> diff --git a/include/linux/devscontrol.h b/include/linux/devscontrol.h
> new file mode 100644
> index 0000000..5f574e0
> --- /dev/null
> +++ b/include/linux/devscontrol.h
> @@ -0,0 +1,20 @@
> +#ifndef __DEVS_CONTROL_H__
> +#define __DEVS_CONTROL_H__
> +struct kobj_map;
> +struct task_struct;
> +
> +#ifdef CONFIG_CGROUP_DEVS
> +struct kobj_map *task_cdev_map(struct task_struct *);
> +struct kobj_map *task_bdev_map(struct task_struct *);
> +#else
> +static inline kobj_map *task_cdev_map(struct task_struct *tsk)
> +{
> + return NULL;
> +}
> +
> +static inline kobj_map *task_bdev_map(struct task_struct *tsk)
> +{
> + return NULL;

> +}
> +#endif
> +#endif
> diff --git a/init/Kconfig b/init/Kconfig
> index b886ac9..4dd53d6 100644
> --- a/init/Kconfig
> +++ b/init/Kconfig
> @@ -283,6 +283,12 @@ config CGROUP_DEBUG
>
>   Say N if unsure
>
> +config CGROUP_DEVS
> + bool "Devices cgroup subsystem"
> + depends on CGROUPS
> + help
> +   Controlls the visibility of devices
> +
> config CGROUP_NS
>   bool "Namespace cgroup subsystem"
>   depends on CGROUPS

_____

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