Subject: [RFC][PATCH] Devices visibility container
Posted by Pavel Emelianov on Mon, 24 Sep 2007 08:28:19 GMT
View Forum Message <> Reply to Message

Hi.

At KS we have pointed out the need in some container, that allows
to limit the visibility of some devices to task within it. I.e.
allow for /dev/null, /dev/zero etc, but disable (by default) some
IDE devices or SCSI discs and so on.

Here's the beta of the container. Currently this only allows to
hide the _character_ devices only from the living tasks. To play
with it you just create the container like this

```
 # mount -t container none /cont/devs -o devices
 # mkdir /cont/devs/0
```

it will have two specific files

```
 # ls /cont/devs
devices.block  devices.char  notify_on_release  releasable  release_agent  tasks
```

then move a task into it

```
 # /bin/echo -n $$ > /cont/devs/0/tasks
```

after this you won't be able to read from even /dev/zero

```
 # hexdump /dev/zero
hexdump: /dev/zero: No such device or address
hexdump: /dev/zero: Bad file descriptor
```

meanwhile from another ssh session you will. You may allow access
to /dev/zero like this

```
 # /bin/echo -n '+1:5' > /cont/devs/0/devices.char
```

More generally, the '+<major>:<minor>' string grants access to
some device, and '-<major>:<minor>' disables one.

The TODO list now looks like this:
* add the block devices support :) don't know how to make it yet;
* make /proc/devices show relevant info depending on who is
  reading it. currently even if major 1 is disabled for task,
  it will be listed in this file;
* make it possible to enable/disable not just individual major:minor
  pair, but something more flexible, e.g. major:* for all minors

```
  for given major or major:m1-m2 for minor range, etc;
* add the ability to restrict the read/write permissions for a
  container. currently one may just control the visible-invisible
  state for a device in a container, but maybe just readable or
  just writable would be better.

This patch is minimally tested, because I just want to know your
opinion on whether it worths developing the container in such a way or not.

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

---

diff --git a/drivers/base/map.c b/drivers/base/map.c
index e87017f..0188053 100644
--- a/drivers/base/map.c
+++ b/drivers/base/map.c
@@ -153,3 +153,21 @@ struct kobj_map *kobj_map_init(kobj_prob
  p->lock = lock;
  return p;
 }
+
+void kobj_map_fini(struct kobj_map *map)
+{
+ int i;
+ struct probe *p, *next;
+
+ for (i = 0; i < 256; i++) {
+  p = map->probes[i];
+  while (p->next != NULL) {
+   next = p->next;
+   kfree(p);
+   p = next;
+  }
+ }
+
+ kfree(p);
+ kfree(map);
+}
diff --git a/fs/Makefile b/fs/Makefile
index 2661ef9..837c731 100644
--- a/fs/Makefile
+++ b/fs/Makefile
@@ -64,6 +64,8 @@ obj-y    += devpts/

 obj-$(CONFIG_PROFILING)  += dcookies.o
 obj-$(CONFIG_DLM)  += dlm/
+
```

```
+obj-$(CONFIG_CONTAINER_DEVS) += devscontrol.o

 # Do not add any filesystems before this line
 obj-$(CONFIG_REISERFS_FS) += reiserfs/
diff --git a/fs/char_dev.c b/fs/char_dev.c
index c3bfa76..1b0e4da 100644
--- a/fs/char_dev.c
+++ b/fs/char_dev.c
@@ -22,6 +22,8 @@
 #include <linux/mutex.h>
 #include <linux/backing-dev.h>

+#include <linux/devscontrol.h>
+
 #ifdef CONFIG_KMOD
 #include <linux/kmod.h>
 #endif
@@ -362,17 +364,24 @@ int chrdev_open(struct inode * inode, st
  struct cdev *p;
  struct cdev *new = NULL;
  int ret = 0;
+ struct kobj_map *map;
+
+ map = task_cdev_map(current);
+ if (map == NULL)
+  map = cdev_map;

  spin_lock(&cdev_lock);
  p = inode->i_cdev;
- if (!p) {
+ if (!p || p->last != map) {
   struct kobject *kobj;
   int idx;
+
   spin_unlock(&cdev_lock);
-  kobj = kobj_lookup(cdev_map, inode->i_rdev, &idx);
+  kobj = kobj_lookup(map, inode->i_rdev, &idx);
   if (!kobj)
    return -ENXIO;
   new = container_of(kobj, struct cdev, kobj);
+  BUG_ON(p != NULL && p != new);
   spin_lock(&cdev_lock);
   p = inode->i_cdev;
   if (!p) {
@@ -384,6 +393,8 @@ int chrdev_open(struct inode * inode, st
    ret = -ENXIO;
  } else if (!cdev_get(p))
   ret = -ENXIO;
```

```
+ if (p)
+  p->last = map;
  spin_unlock(&cdev_lock);
  cdev_put(new);
  if (ret)
@@ -461,6 +472,49 @@ int cdev_add(struct cdev *p, dev_t dev,
  return kobj_map(cdev_map, dev, count, NULL, exact_match, exact_lock, p);
 }

+int cdev_add_to_map(struct kobj_map *map, dev_t dev)
+{
+ int tmp;
+ struct kobject *k;
+ struct cdev *c;
+
+ k = kobj_lookup(cdev_map, dev, &tmp);
+ if (k == NULL)
+  return -ENODEV;
+
+ c = container_of(k, struct cdev, kobj);
+ tmp = kobj_map(map, dev, 1, NULL, exact_match, exact_lock, c);
+ if (tmp < 0) {
+  cdev_put(c);
+  return tmp;
+ }
+
+ return 0;
+}
+
+int cdev_del_from_map(struct kobj_map *map, dev_t dev)
+{
+ int tmp;
+ struct kobject *k;
+ struct cdev *c;
+
+ k = kobj_lookup(cdev_map, dev, &tmp);
+ if (k == NULL)
+  return -ENODEV;
+
+ c = container_of(k, struct cdev, kobj);
+ kobj_unmap(map, dev, 1);
+
+ spin_lock(&cdev_lock);
+ if (c->last == map)
+  c->last = NULL;
+ spin_unlock(&cdev_lock);
+
+ cdev_put(c);
```

```
+ cdev_put(c);
+ return 0;
+}
+
 static void cdev_unmap(dev_t dev, unsigned count)
 {
  kobj_unmap(cdev_map, dev, count);
@@ -542,6 +596,16 @@ static struct kobject *base_probe(dev_t
  return NULL;
 }

+struct kobj_map *cdev_map_init(void)
+{
+ return kobj_map_init(base_probe, &chrdevs_lock);
+}
+
+void cdev_map_fini(struct kobj_map *map)
+{
+ kobj_map_fini(map);
+}
+
 void __init chrdev_init(void)
 {
  cdev_map = kobj_map_init(base_probe, &chrdevs_lock);
diff --git a/fs/devscontrol.c b/fs/devscontrol.c
new file mode 100644
index 0000000..6fb5f05
--- /dev/null
+++ b/fs/devscontrol.c
@@ -0,0 +1,170 @@
+/*
+ * devscontrol.c - Device Controller
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ * Author: Pavel Emelyanov <xemul@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/container.h>
```

```
+#include <linux/cdev.h>
+#include <linux/err.h>
+#include <linux/devscontrol.h>
+#include <linux/uaccess.h>
+
+struct devs_container {
+ struct container_subsys_state css;
+
+ struct kobj_map *cdev_map;
+};
+
+static inline
+struct devs_container *css_to_devs(struct container_subsys_state *css)
+{
+ return container_of(css, struct devs_container, css);
+}
+
+static inline
+struct devs_container *container_to_devs(struct container *cont)
+{
+ return css_to_devs(container_subsys_state(cont, devs_subsys_id));
+}
+
+struct kobj_map *task_cdev_map(struct task_struct *tsk)
+{
+ struct container_subsys_state *css;
+
+ css = task_subsys_state(tsk, devs_subsys_id);
+ if (css->container->parent == NULL)
+  return NULL;
+ else
+  return css_to_devs(css)->cdev_map;
+}
+
+static struct container_subsys_state *
+devs_create(struct container_subsys *ss, struct container *cont)
+{
+ struct devs_container *devs;
+
+ devs = kzalloc(sizeof(struct devs_container), GFP_KERNEL);
+ if (devs == NULL)
+  goto out;
+
+ devs->cdev_map = cdev_map_init();
+ if (devs->cdev_map == NULL)
+  goto out_free;
+
+ return &devs->css;
```

```
+
+out_free:
+ kfree(devs);
+out:
+ return ERR_PTR(-ENOMEM);
+}
+
+static void devs_destroy(struct container_subsys *ss, struct container *cont)
+{
+ struct devs_container *devs;
+
+ devs = container_to_devs(cont);
+ cdev_map_fini(devs->cdev_map);
+ kfree(devs);
+}
+
+static int decode_dev_name(char *buf, dev_t *dev)
+{
+ unsigned int major, minor;
+ char *end;
+
+ major = simple_strtoul(buf, &end, 10);
+ if (*end != ':')
+  return -EINVAL;
+
+ minor = simple_strtoul(end + 1, &end, 10);
+ if (*end != '\0')
+  return -EINVAL;
+
+ *dev = MKDEV(major, minor);
+ return 0;
+}
+
+static ssize_t
+devs_char_write(struct container *cont, struct cftype *cft, struct file *f,
+  const char __user *userbuf, size_t nbytes, loff_t *pos)
+{
+ int err;
+ dev_t dev;
+ char buf[64];
+ struct devs_container *devs;
+
+ if (copy_from_user(buf, userbuf, sizeof(buf)))
+  return -EFAULT;
+
+ err = decode_dev_name(buf + 1, &dev);
+ if (err < 0)
+  return err;
```

```
+
+ devs = container_to_devs(cont);
+
+ switch (buf[0]) {
+ case '+':
+  err = cdev_add_to_map(devs->cdev_map, dev);
+  if (err < 0)
+   return err;
+
+  css_get(&devs->css);
+  break;
+ case '-':
+  err = cdev_del_from_map(devs->cdev_map, dev);
+  if (err < 0)
+   return err;
+
+  css_put(&devs->css);
+  break;
+ default:
+  return -EINVAL;
+ }
+
+ return nbytes;
+}
+
+static ssize_t
+devs_block_write(struct container *cont, struct cftype *cft, struct file *f,
+  const char __user *userbuf, size_t nbytes, loff_t *pos)
+{
+ return -ENOTTY;
+}
+
+static struct cftype devs_files[] = {
+ {
+  .name = "char",
+  .write = devs_char_write,
+ },
+ {
+  .name = "block",
+  .write = devs_block_write,
+ },
+};
+
+static int devs_populate(struct container_subsys *ss, struct container *cont)
+{
+ return container_add_files(cont, ss,
+   devs_files, ARRAY_SIZE(devs_files));
+}
```

```
+
+struct container_subsys devs_subsys = {
+ .name = "devices",
+ .subsys_id = devs_subsys_id,
+ .create = devs_create,
+ .destroy = devs_destroy,
+ .populate = devs_populate,
+};
diff --git a/include/linux/cdev.h b/include/linux/cdev.h
index 1e29b13..0eded40 100644
--- a/include/linux/cdev.h
+++ b/include/linux/cdev.h
@@ -9,6 +9,7 @@
 struct file_operations;
 struct inode;
 struct module;
+struct kobj_map;

 struct cdev {
  struct kobject kobj;
@@ -17,6 +18,7 @@ struct cdev {
  struct list_head list;
  dev_t dev;
  unsigned int count;
+ struct kobj_map *last;
 };

 void cdev_init(struct cdev *, const struct file_operations *);
@@ -33,5 +35,9 @@ void cd_forget(struct inode *);

 extern struct backing_dev_info directly_mappable_cdev_bdi;

+int cdev_add_to_map(struct kobj_map *map, dev_t dev);
+int cdev_del_from_map(struct kobj_map *map, dev_t dev);
+struct kobj_map *cdev_map_init(void);
+void cdev_map_fini(struct kobj_map *map);
 #endif
 #endif
diff --git a/include/linux/container_subsys.h b/include/linux/container_subsys.h
index 81d11c2..9315a9b 100644
--- a/include/linux/container_subsys.h
+++ b/include/linux/container_subsys.h
@@ -36,3 +36,9 @@ SUBSYS(mem_container)
 #endif

 /* */
+
+#ifdef CONFIG_CONTAINER_DEVS
```

```
+SUBSYS(devs)
+#endif
+
+/* */
diff --git a/include/linux/devscontrol.h b/include/linux/devscontrol.h
new file mode 100644
index 0000000..51ae916
--- /dev/null
+++ b/include/linux/devscontrol.h
@@ -0,0 +1,14 @@
+#ifndef __DEVS_CONTROL_H__
+#define __DEVS_CONTROL_H__
+struct kobj_map;
+struct task_struct;
+
+#ifdef CONFIG_CONTAINER_DEVS
+struct kobj_map *task_cdev_map(struct task_struct *);
+#else
+static inline kobj_map *task_cdev_map(struct task_struct *tsk)
+{
+ return NULL;
+}
+#endif
+#endif
diff --git a/include/linux/kobj_map.h b/include/linux/kobj_map.h
index bafe178..2476f8d 100644
--- a/include/linux/kobj_map.h
+++ b/include/linux/kobj_map.h
@@ -10,5 +10,6 @@ int kobj_map(struct kobj_map *, dev_t, u
 void kobj_unmap(struct kobj_map *, dev_t, unsigned long);
 struct kobject *kobj_lookup(struct kobj_map *, dev_t, int *);
 struct kobj_map *kobj_map_init(kobj_probe_t *, struct mutex *);
+void kobj_map_fini(struct kobj_map *);

 #endif
diff --git a/init/Kconfig b/init/Kconfig
index 0bb211a..5e1158e 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -292,6 +292,12 @@ config CONTAINER_DEBUG

    Say N if unsure

+config CONTAINER_DEVS
+ bool "Devices container subsystem"
+ depends on CONTAINERS
+ help
+   Controlls the visibility of devices
```

+
 config CONTAINER_NS
        bool "Namespace container subsystem"
        depends on CONTAINERS

_____

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