Subject: [PATCH 0/4] Devices accessibility control group (v3, release candidate)
Posted by Pavel Emelianov on Thu, 07 Feb 2008 12:56:18 GMT
View Forum Message <> Reply to Message

Changes from v2:
* Fixed problems pointed out by Sukadev with permissions
  revoke. Now we have to perform kobject re-lookup on
  each char device open, just like for block ones, so I
  think this is OK.

The /proc/devices tune is still in TODO list, as I have
problems with getting majors _in_a_simple_manner_ from a
map, that contains a mix of major/minor pairs in
arbitrary order.

The second version is here:
http://openvz.org/pipermail/devel/2008-January/010160.html
Changes from v1:

* Added the block devices support :) It turned out to
  be a bit simpler than the char one (or I missed
  something significant);
* Now we can enable/disable not just individual devices,
  but the whole major with all its minors (see the TODO
  list beyond as well);
* Added the ability to restrict the read/write permissions
  to devices, not just visible/invisible state.

The first version was here:
http://openvz.org/pipermail/devel/2007-September/007647.html

I still don't pay much attention to split this set well, so
this will most likely won't work with git-bisect, but I think
this is OK for now. I will sure split it better when I send
it to Andrew.

The set is prepared against the 2.6.24-rc8-mm1.

To play with it - run a standard procedure:

```
 # mount -t container none /cont/devs -o devices
 # mkdir /cont/devs/0
 # echo -n $$ > /cont/devs/0/tasks
```

and tune device permissions.

Thanks,
Pavel

Subject: [PATCH 1/4] Some changes in the kobject mapper
Posted by Pavel Emelianov on Thu, 07 Feb 2008 12:57:23 GMT
View Forum Message <> Reply to Message

The main thing that I want from the kobj mapper is to add
the mode_t on the struct kobj_map that reflects with
permissions are associated with this particular map. This
mode is to be returned via the kobj_lookup.

I use the FMODE_XXX flags to handle the permissions bits,
as I will compare these ones to the file->f_mode later.
By default all bits are set (for the initial container).

The additional things I need are kobj_remap() to change
that permission and kobj_iterate() to walk the map.

The kobj_map_fini() is the roll-back for the kobj_map_init().

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

```
diff --git a/drivers/base/map.c b/drivers/base/map.c
index e87017f..1aa2b58 100644
--- a/drivers/base/map.c
+++ b/drivers/base/map.c
@@ -15,11 +15,13 @@
 #include <linux/kdev_t.h>
 #include <linux/kobject.h>
 #include <linux/kobj_map.h>
+#include <linux/fs.h>

 struct kobj_map {
  struct probe {
   struct probe *next;
   dev_t dev;
+  mode_t mode;
   unsigned long range;
   struct module *owner;
   kobj_probe_t *get;
@@ -29,9 +31,9 @@ struct kobj_map {
  struct mutex *lock;
```

```
 };

-int kobj_map(struct kobj_map *domain, dev_t dev, unsigned long range,
-      struct module *module, kobj_probe_t *probe,
-      int (*lock)(dev_t, void *), void *data)
+static int __kobj_map(struct kobj_map *domain, dev_t dev, mode_t mode,
+  unsigned long range, struct module *module,
+  kobj_probe_t *probe, int (*lock)(dev_t, void *), void *data)
 {
  unsigned n = MAJOR(dev + range - 1) - MAJOR(dev) + 1;
  unsigned index = MAJOR(dev);
@@ -53,8 +55,10 @@ int kobj_map(struct kobj_map *domain, dev_t dev, unsigned long range,
   p->dev = dev;
   p->range = range;
   p->data = data;
+  /* we allow these ones always by default */
+  p->mode = mode | FMODE_LSEEK | FMODE_PREAD | FMODE_PWRITE;
  }
- mutex_lock(domain->lock);
+
  for (i = 0, p -= n; i < n; i++, p++, index++) {
   struct probe **s = &domain->probes[index % 255];
   while (*s && (*s)->range < range)
@@ -62,10 +66,57 @@ int kobj_map(struct kobj_map *domain, dev_t dev, unsigned long range,
   p->next = *s;
   *s = p;
  }
- mutex_unlock(domain->lock);
  return 0;
 }

+int kobj_map(struct kobj_map *domain, dev_t dev, unsigned long range,
+      struct module *module, kobj_probe_t *probe,
+      int (*lock)(dev_t, void *), void *data)
+{
+ int err;
+
+ mutex_lock(domain->lock);
+ err = __kobj_map(domain, dev, FMODE_READ | FMODE_WRITE, range,
+   module, probe, lock, data);
+ mutex_unlock(domain->lock);
+ return err;
+}
+
+#ifdef CONFIG_CGROUP_DEVS
+int kobj_remap(struct kobj_map *domain, dev_t dev, mode_t mode,
+  unsigned long range, struct module *module,
+  kobj_probe_t *probe, int (*lock)(dev_t, void *), void *data)
```

```
+{
+ unsigned n = MAJOR(dev + range - 1) - MAJOR(dev) + 1;
+ unsigned index = MAJOR(dev);
+ unsigned i;
+ int err = -ESRCH;
+
+ if (n > 255)
+  n = 255;
+
+ mutex_lock(domain->lock);
+ for (i = 0; i < n; i++, index++) {
+  struct probe **s;
+  for (s = &domain->probes[index % 255]; *s; s = &(*s)->next) {
+   struct probe *p = *s;
+   if (p->dev == dev) {
+    p->mode = mode | FMODE_LSEEK |
+     FMODE_PREAD | FMODE_PWRITE;
+    err = 0;
+    break;
+   }
+  }
+ }
+
+ if (err)
+  err = __kobj_map(domain, dev, mode, range, module,
+    probe, lock, data);
+ mutex_unlock(domain->lock);
+ return err;
+}
+#endif
+
 void kobj_unmap(struct kobj_map *domain, dev_t dev, unsigned long range)
 {
  unsigned n = MAJOR(dev + range - 1) - MAJOR(dev) + 1;
@@ -93,7 +144,8 @@ void kobj_unmap(struct kobj_map *domain, dev_t dev, unsigned long
range)
  kfree(found);
 }

-struct kobject *kobj_lookup(struct kobj_map *domain, dev_t dev, int *index)
+struct kobject *kobj_lookup(struct kobj_map *domain, dev_t dev, mode_t *mode,
+  int *index)
 {
  struct kobject *kobj;
  struct probe *p;
@@ -125,14 +177,46 @@ retry:
   kobj = probe(dev, index, data);
   /* Currently ->owner protects _only_ ->probe() itself. */
```

```
   module_put(owner);
-  if (kobj)
+  if (kobj) {
+   if (mode)
+    *mode = p->mode;
    return kobj;
+  }
   goto retry;
  }
  mutex_unlock(domain->lock);
  return NULL;
 }

+#ifdef CONFIG_CGROUP_DEVS
+void kobj_map_iterate(struct kobj_map *domain,
+  int (*fn)(dev_t, int, mode_t, void *), void *arg)
+{
+ int i;
+ struct probe *p;
+ dev_t skip = MKDEV(0, 0);
+
+ mutex_lock(domain->lock);
+ for (i = 0; i < 255; i++) {
+  p = domain->probes[i];
+  while (p != NULL) {
+   if (p->dev == skip)
+    goto next;
+   if (p->dev == MKDEV(0, 1))
+    goto next;
+
+   skip = p->dev;
+   if (fn(p->dev, p->range, p->mode, arg))
+    goto done;
+next:
+   p = p->next;
+  }
+ }
+done:
+ mutex_unlock(domain->lock);
+}
+#endif
+
 struct kobj_map *kobj_map_init(kobj_probe_t *base_probe, struct mutex *lock)
 {
  struct kobj_map *p = kmalloc(sizeof(struct kobj_map), GFP_KERNEL);
@@ -153,3 +237,21 @@ struct kobj_map *kobj_map_init(kobj_probe_t *base_probe, struct
mutex *lock)
  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/include/linux/kobj_map.h b/include/linux/kobj_map.h
index bafe178..ecfe772 100644
--- a/include/linux/kobj_map.h
+++ b/include/linux/kobj_map.h
@@ -7,8 +7,13 @@ struct kobj_map;

 int kobj_map(struct kobj_map *, dev_t, unsigned long, struct module *,
       kobj_probe_t *, int (*)(dev_t, void *), void *);
+int kobj_remap(struct kobj_map *, dev_t, mode_t, unsigned long, struct module *,
+      kobj_probe_t *, int (*)(dev_t, void *), void *);
 void kobj_unmap(struct kobj_map *, dev_t, unsigned long);
-struct kobject *kobj_lookup(struct kobj_map *, dev_t, int *);
+struct kobject *kobj_lookup(struct kobj_map *, dev_t, mode_t *, int *);
+void kobj_map_iterate(struct kobj_map *, int (*fn)(dev_t, int, mode_t, void *),
+  void *);
 struct kobj_map *kobj_map_init(kobj_probe_t *, struct mutex *);
+void kobj_map_fini(struct kobj_map *);

 #endif
```

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

---

Subject: [PATCH 2/4] The character devices layer changes

These changes include the API for the control group
to map/remap/unmap the devices with their permissions
and one important thing.

The fact is that the struct cdev is cached in the inode
for faster access, so once we looked one up we go through
the fast path and omit the kobj_lookup() call. This is no
longer good when we restrict the access to cdevs. Another
problem is that different char devices may use one struct
cdev object, so having an access to one of them grants us
access to another, so we have to re-lookup the kobj map
each open.

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

```
diff --git a/fs/char_dev.c b/fs/char_dev.c
index 2c7a8b5..ddacab7 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 +364,31 @@ int chrdev_open(struct inode * inode, struct file * filp)
  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 || map != cdev_map) {
   struct kobject *kobj;
   int idx;
```

```
+  mode_t mode;
+
   spin_unlock(&cdev_lock);
-  kobj = kobj_lookup(cdev_map, inode->i_rdev, &idx);
+  kobj = kobj_lookup(map, inode->i_rdev, &mode, &idx);
   if (!kobj)
    return -ENXIO;
   new = container_of(kobj, struct cdev, kobj);
+  BUG_ON(p != NULL && p != new);
+
+  if ((filp->f_mode & mode) != filp->f_mode) {
+   cdev_put(new);
+   return -EACCES;
+  }
+
   spin_lock(&cdev_lock);
   p = inode->i_cdev;
   if (!p) {
@@ -461,6 +477,53 @@ int cdev_add(struct cdev *p, dev_t dev, unsigned count)
  return kobj_map(cdev_map, dev, count, NULL, exact_match, exact_lock, p);
 }

+#ifdef CONFIG_CGROUP_DEVS
+int cdev_add_to_map(struct kobj_map *map, dev_t dev, int all, mode_t mode)
+{
+ int tmp;
+ struct kobject *k;
+ struct cdev *c;
+
+ k = kobj_lookup(cdev_map, dev, NULL, &tmp);
+ if (k == NULL)
+  return -ENODEV;
+
+ c = container_of(k, struct cdev, kobj);
+ tmp = kobj_remap(map, dev, mode, all ? MINORMASK : 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 all)
+{
+ int tmp;
+ struct kobject *k;
```

```
+ struct cdev *c;
+
+ k = kobj_lookup(cdev_map, dev, NULL, &tmp);
+ if (k == NULL)
+  return -ENODEV;
+
+ c = container_of(k, struct cdev, kobj);
+ kobj_unmap(map, dev, all ? MINORMASK : 1);
+
+ cdev_put(c);
+ cdev_put(c);
+ return 0;
+}
+
+void cdev_iterate_map(struct kobj_map *map,
+  int (*fn)(dev_t, int, mode_t, void *), void *x)
+{
+ kobj_map_iterate(map, fn, x);
+}
+#endif
+
 static void cdev_unmap(dev_t dev, unsigned count)
 {
  kobj_unmap(cdev_map, dev, count);
@@ -540,9 +603,19 @@ static struct kobject *base_probe(dev_t dev, int *part, void *data)
  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);
+ cdev_map = cdev_map_init();
  bdi_init(&directly_mappable_cdev_bdi);
 }

diff --git a/include/linux/cdev.h b/include/linux/cdev.h
index 1e29b13..fe0e560 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;
@@ -33,5 +34,11 @@ 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 all, mode_t mode);
+int cdev_del_from_map(struct kobj_map *map, dev_t dev, int all);
+struct kobj_map *cdev_map_init(void);
+void cdev_map_fini(struct kobj_map *map);
+void cdev_iterate_map(struct kobj_map *,
+  int (*fn)(dev_t, int, mode_t, void *), void *);
 #endif
 #endif
```

_____

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

---

Subject: [PATCH 3/4] The block devices layer changes
Posted by Pavel Emelianov on Thu, 07 Feb 2008 12:59:48 GMT
View Forum Message <> Reply to Message

They are the same as for the character layer, but
the good news is that there are no caching in this
case.

So this patch is smaller and easier to understand
as compared to the previous one.

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

```
diff --git a/block/genhd.c b/block/genhd.c
index 5e4ab4b..6f9ef48 100644
--- a/block/genhd.c
+++ b/block/genhd.c
@@ -8,6 +8,7 @@
```

```
 #include <linux/kdev_t.h>
 #include <linux/kernel.h>
 #include <linux/blkdev.h>
+#include <linux/devscontrol.h>
 #include <linux/init.h>
 #include <linux/spinlock.h>
 #include <linux/seq_file.h>
@@ -195,6 +196,57 @@ void unlink_gendisk(struct gendisk *disk)
         disk->minors);
 }

+#ifdef CONFIG_CGROUP_DEVS
+int bdev_add_to_map(struct kobj_map *map, dev_t dev, int all, mode_t mode)
+{
+ int tmp;
+ struct kobject *kobj;
+ struct device *d;
+ struct gendisk *disk;
+
+ kobj = kobj_lookup(bdev_map, dev, NULL, &tmp);
+ if (kobj == NULL)
+  return -ENODEV;
+
+ d = kobj_to_dev(kobj);
+ disk = dev_to_disk(d);
+ tmp = kobj_remap(map, dev, mode, all ? MINORBITS : 1, NULL,
+   exact_match, exact_lock, disk);
+ if (tmp < 0) {
+  put_disk(disk);
+  return tmp;
+ }
+
+ return 0;
+}
+
+int bdev_del_from_map(struct kobj_map *map, dev_t dev, int all)
+{
+ int tmp;
+ struct kobject *kobj;
+ struct device *d;
+ struct gendisk *disk;
+
+ kobj = kobj_lookup(bdev_map, dev, NULL, &tmp);
+ if (kobj == NULL)
+  return -ENODEV;
+
+ d = kobj_to_dev(kobj);
+ disk = dev_to_disk(d);
```

```
+ kobj_unmap(map, dev, all ? MINORBITS : 1);
+
+ put_disk(disk);
+ put_disk(disk);
+ return 0;
+}
+
+void bdev_iterate_map(struct kobj_map *map,
+  int (*fn)(dev_t, int, mode_t, void *), void *x)
+{
+ kobj_map_iterate(map, fn, x);
+}
+#endif
+
 /**
  * get_gendisk - get partitioning information for a given device
  * @dev: device to get partitioning information for
@@ -202,10 +254,18 @@ void unlink_gendisk(struct gendisk *disk)
  * This function gets the structure containing partitioning
  * information for the given device @dev.
  */
-struct gendisk *get_gendisk(dev_t devt, int *part)
+struct gendisk *get_gendisk(dev_t devt, mode_t *mode, int *part)
 {
- struct kobject *kobj = kobj_lookup(bdev_map, devt, part);
- struct device *dev = kobj_to_dev(kobj);
+ struct kobj_map *map;
+ struct kobject *kobj;
+ struct device *dev;
+
+ map = task_bdev_map(current);
+ if (map == NULL)
+  map = bdev_map;
+
+ kobj = kobj_lookup(map, devt, mode, part);
+ dev = kobj_to_dev(kobj);

  return  kobj ? dev_to_disk(dev) : NULL;
 }
@@ -356,10 +416,20 @@ static struct kobject *base_probe(dev_t devt, int *part, void *data)
  return NULL;
 }

+struct kobj_map *bdev_map_init(void)
+{
+ return kobj_map_init(base_probe, &block_class_lock);
+}
+
```

```
+void bdev_map_fini(struct kobj_map *map)
+{
+ kobj_map_fini(map);
+}
+
 static int __init genhd_device_init(void)
 {
  class_register(&block_class);
- bdev_map = kobj_map_init(base_probe, &block_class_lock);
+ bdev_map = bdev_map_init();
  blk_dev_init();

 #ifndef CONFIG_SYSFS_DEPRECATED
diff --git a/fs/block_dev.c b/fs/block_dev.c
index 55295a4..03b1b5e 100644
--- a/fs/block_dev.c
+++ b/fs/block_dev.c
@@ -1129,16 +1129,25 @@ static int do_open(struct block_device *bdev, struct file *file, int
for_part)
  struct module *owner = NULL;
  struct gendisk *disk;
  int ret = -ENXIO;
+ mode_t mode;
  int part;

  file->f_mapping = bdev->bd_inode->i_mapping;
  lock_kernel();
- disk = get_gendisk(bdev->bd_dev, &part);
+ disk = get_gendisk(bdev->bd_dev, &mode, &part);
  if (!disk) {
   unlock_kernel();
   bdput(bdev);
   return ret;
  }
+
+ if ((file->f_mode & mode) != file->f_mode) {
+  unlock_kernel();
+  bdput(bdev);
+  put_disk(disk);
+  return -EACCES;
+ }
+
  owner = disk->fops->owner;

  mutex_lock_nested(&bdev->bd_mutex, for_part);
diff --git a/include/linux/genhd.h b/include/linux/genhd.h
index 1dbea0a..4a92b65 100644
--- a/include/linux/genhd.h
```

```
+++ b/include/linux/genhd.h
@@ -240,7 +240,15 @@ extern int get_blkdev_list(char *, int);
 extern void add_disk(struct gendisk *disk);
 extern void del_gendisk(struct gendisk *gp);
 extern void unlink_gendisk(struct gendisk *gp);
-extern struct gendisk *get_gendisk(dev_t dev, int *part);
+extern struct gendisk *get_gendisk(dev_t dev, mode_t *mode, int *part);
+
+struct kobj_map;
+extern int bdev_add_to_map(struct kobj_map *, dev_t dev, int all, mode_t mode);
+extern int bdev_del_from_map(struct kobj_map *map, dev_t dev, int all);
+extern void bdev_iterate_map(struct kobj_map *map,
+  int (*fn)(dev_t, int, mode_t, void *), void *x);
+extern struct kobj_map *bdev_map_init(void);
+extern void bdev_map_fini(struct kobj_map *map);

 extern void set_device_ro(struct block_device *bdev, int flag);
 extern void set_disk_ro(struct gendisk *disk, int flag);
```

Subject: [PATCH 4/4] The control group itself
Posted by Pavel Emelianov on Thu, 07 Feb 2008 13:01:47 GMT
View Forum Message <> Reply to Message

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.

The top container isn't initialized, so that the char
and block layers will use the global maps to lookup
their devices. I did that not to export the static maps
to the outer world.

Good news is that this patch now contains more comments
and Documentation file :)

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

---

diff --git a/Documentation/controllers/devices.txt b/Documentation/controllers/devices.txt
new file mode 100644

```
index 0000000..dbd0c7a
--- /dev/null
+++ b/Documentation/controllers/devices.txt
@@ -0,0 +1,61 @@
+
+ Devices visibility controller
+
+This controller allows to tune the devices accessibility by tasks,
+i.e. grant full access for /dev/null, /dev/zero etc, grant read-only
+access to IDE devices and completely hide SCSI disks.
+
+Tasks still can call mknod to create device files, regardless of
+whether the particular device is visible or accessible, but they
+may not be able to open it later.
+
+This one hides under CONFIG_CGROUP_DEVS option.
+
+
+Configuring
+
+The controller provides a single file to configure itself -- the
+devices.permissions one. To change the accessibility level for some
+device write the following string into it:
+
+[cb] <major>:(<minor>|*) [r-][w-]
+ ^         ^             ^
+ |         |             |
+ |         |             +--- access rights (1)
+ |         |
+ |         +-- device major and minor numbers (2)
+ |
+ +-- device type (character / block)
+
+1) The access rights set to '--' remove the device from the group's
+access list, so that it will not even be shown in this file later.
+
+2) Setting the minor to '*' grants access to all the minors for
+particular major.
+
+When reading from it, one may see something like
+
+ c 1:5 rw
+ b 8:* r-
+
+Security issues, concerning who may grant access to what are governed
+at the cgroup infrastructure level.
+
+
```

```
+Examples:
+
+1. Grand full access to /dev/null
+ # echo c 1:3 rw > /cgroups/<id>/devices.permissions
+
+2. Grant the read-only access to /dev/sda and partitions
+ # echo b 8:* r- > ...
+
+3. Change the /dev/null access to write-only
+ # echo c 1:3 -w > ...
+
+4. Revoke access to /dev/sda
+ # echo b 8:* -- > ...
+
+
+  Written by Pavel Emelyanov <xemul@openvz.org>
+
diff --git a/fs/Makefile b/fs/Makefile
index 7996220..5ad03be 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_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..48c5f69
--- /dev/null
+++ b/fs/devscontrol.c
@@ -0,0 +1,314 @@
+/*
+ * devscontrol.c - Device Controller
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ * Author: Pavel Emelyanov <xemul at openvz dot 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 {
+ /*
+  * The subsys state to build into cgrous infrastructure
+  */
+ struct cgroup_subsys_state css;
+
+ /*
+  * The maps of character and block devices. They provide a
+  * map from dev_t-s to struct cdev/gendisk. See fs/char_dev.c
+  * and block/genhd.c to find out how the ->open() callbacks
+  * work when opening a device.
+  *
+  * Each group will have its onw maps, and at the open()
+  * time code will lookup in this map to get the device
+  * and permissions by its dev_t.
+  */
+ 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);
+}
+
+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);
+}
+
+/*
+ * The devices.permissions file read/write functionality
+ *
+ * The following two routines parse and print the strings like
+ * [cb] <major>:(<minor>|*) [r-][w-]
+ */
+
+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;
+
+ 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;
+
+ 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);
+
+ /*
+  * No locking here is required - all that we need
+  * is provided inside the kobject mapping code
+  */
+
+ 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..38057b9
--- /dev/null
+++ b/include/linux/devscontrol.h
@@ -0,0 +1,26 @@
+#ifndef __DEVS_CONTROL_H__
+#define __DEVS_CONTROL_H__
+struct kobj_map;
+struct task_struct;
+
+/*
+ * task_[cb]dev_map - get a map from task. Both calls may return
+ * NULL, to indicate, that task doesn't belong to any group and
+ * that the global map is to be used.
+ */
+
+#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 732a1c2..f9a1b4f 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -283,6 +283,19 @@ config CGROUP_DEBUG

    Say N if unsure

+config CGROUP_DEVS
+ bool "Devices cgroup subsystem"
+ depends on CGROUPS
+ help
+   Controlls the access rights to devices, i.e. you may hide
+   some of them from tasks, so that they will not be able
+   to open them, or you may grant a read-only access to some
+   of them.
+
+   See Documentation/controllers/devices.txt for details.
+
+   This is harmless to say N here, so do it if unsure.
+
 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

---

Subject: Re: [PATCH 0/4] Devices accessibility control group (v3, release candidate)

Posted by serue on Fri, 08 Feb 2008 16:12:17 GMT

View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):

> Changes from v2:
> * Fixed problems pointed out by Sukadev with permissions
>   revoke. Now we have to perform kobject re-lookup on
>   each char device open, just like for block ones, so I
>   think this is OK.
>
> The /proc/devices tune is still in TODO list, as I have
> problems with getting majors _in_a_simple_manner_ from a
> map, that contains a mix of major/minor pairs in
> arbitrary order.

Thanks for posting, Pavel. I'm sorry, I'm *trying* to review these, but
my eyes are glazing over. (Not because of your patches I'll start by
just testing it a bit either this afternoon or monday.

thanks,
-serge

> The second version is here:
> http://openvz.org/pipermail/devel/2008-January/010160.html
> Changes from v1:
>
> * Added the block devices support :) It turned out to
>   be a bit simpler than the char one (or I missed
>   something significant);
> * Now we can enable/disable not just individual devices,
>   but the whole major with all its minors (see the TODO
>   list beyond as well);
> * Added the ability to restrict the read/write permissions
>   to devices, not just visible/invisible state.
>
> The first version was here:
> http://openvz.org/pipermail/devel/2007-September/007647.html
>
> I still don't pay much attention to split this set well, so
> this will most likely won't work with git-bisect, but I think
> this is OK for now. I will sure split it better when I send
> it to Andrew.
>
> The set is prepared against the 2.6.24-rc8-mm1.
>
> To play with it - run a standard procedure:
>
> # mount -t container none /cont/devs -o devices
> # mkdir /cont/devs/0
> # echo -n $$ > /cont/devs/0/tasks
>
> and tune device permissions.

>
> Thanks,
> Pavel

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

---

Subject: Re: [PATCH 4/4] The control group itself
Posted by serue on Mon, 11 Feb 2008 17:38:30 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.
>
> The top container isn't initialized, so that the char
> and block layers will use the global maps to lookup
> their devices. I did that not to export the static maps
> to the outer world.
>
> Good news is that this patch now contains more comments
> and Documentation file :)

Seems to work as advertised :) I can't reproduce Suka's null/zero
bug.

You're relying fully on uid-0 to stop writes into the
devices.permissions files. Would you mind adding a check for
CAP_SYS_ADMIN (or CAP_NS_OVERRIDE+CAP_MKNOD)? Or were you really
counting on using the filesystem visibility cgroup to stop a container
from making changes to its device access whitelist?

thanks,
-serge

> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> ---
>
> diff --git a/Documentation/controllers/devices.txt b/Documentation/controllers/devices.txt
> new file mode 100644
> index 0000000..dbd0c7a
> --- /dev/null
> +++ b/Documentation/controllers/devices.txt

> @@ -0,0 +1,61 @@
> +
> + Devices visibility controller
> +
> +This controller allows to tune the devices accessibility by tasks,
> +i.e. grant full access for /dev/null, /dev/zero etc, grant read-only
> +access to IDE devices and completely hide SCSI disks.
> +
> +Tasks still can call mknod to create device files, regardless of
> +whether the particular device is visible or accessible, but they
> +may not be able to open it later.
> +
> +This one hides under CONFIG_CGROUP_DEVS option.
> +
> +
> +Configuring
> +
> +The controller provides a single file to configure itself -- the
> +devices.permissions one. To change the accessibility level for some
> +device write the following string into it:
> +
> +[cb] <major>:(<minor>|*) [r-][w-]
> + ^   ^   ^
> + | | |
> + | | +--- access rights (1)
> + | |
> + | +-- device major and minor numbers (2)
> + |
> + +-- device type (character / block)
> +
> +1) The access rights set to '--' remove the device from the group's
> +access list, so that it will not even be shown in this file later.
> +
> +2) Setting the minor to '*' grants access to all the minors for
> +particular major.
> +
> +When reading from it, one may see something like
> +
> + c 1:5 rw
> + b 8:* r-
> +
> +Security issues, concerning who may grant access to what are governed
> +at the cgroup infrastructure level.
> +
> +
> +Examples:
> +
> +1. Grand full access to /dev/null

> + # echo c 1:3 rw > /cgroups/<id>/devices.permissions
> +
> +2. Grant the read-only access to /dev/sda and partitions
> + # echo b 8:* r- > ...
> +
> +3. Change the /dev/null access to write-only
> + # echo c 1:3 -w > ...
> +
> +4. Revoke access to /dev/sda
> + # echo b 8:* -- > ...
> +
> +
> + Written by Pavel Emelyanov <xemul@openvz.org>
> +
> diff --git a/fs/Makefile b/fs/Makefile
> index 7996220..5ad03be 100644
> --- a/fs/Makefile
> +++ b/fs/Makefile
> @@ -64,6 +64,8 @@ obj-y   += devpts/
>
> obj-$(CONFIG_PROFILING) += dcookies.o
> obj-$(CONFIG_DLM) += dlm/
> +
> +obj-$(CONFIG_CGROUP_DEVS) += devscontrol.o
>
> # Do not add any filesystems before this line
> obj-$(CONFIG_REISERFS_FS) += reiserfs/
> diff --git a/fs/devscontrol.c b/fs/devscontrol.c
> new file mode 100644
> index 0000000..48c5f69
> --- /dev/null
> +++ b/fs/devscontrol.c
> @@ -0,0 +1,314 @@
> +/*
> + * devscontrol.c - Device Controller
> + *
> + * Copyright 2007 OpenVZ SWsoft Inc
> + * Author: Pavel Emelyanov <xemul at openvz dot 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 {
> + /*
> +  * The subsys state to build into cgrous infrastructure
> +  */
> + struct cgroup_subsys_state css;
> +
> + /*
> +  * The maps of character and block devices. They provide a
> +  * map from dev_t-s to struct cdev/gendisk. See fs/char_dev.c
> +  * and block/genhd.c to find out how the ->open() callbacks
> +  * work when opening a device.
> +  *
> +  * Each group will have its onw maps, and at the open()
> +  * time code will lookup in this map to get the device
> +  * and permissions by its dev_t.
> +  */
> + 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);
> +}
> +
> +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);
> +}
> +
> +/*
> + * The devices.permissions file read/write functionality
> + *
> + * The following two routines parse and print the strings like
> + * [cb] <major>:(<minor>|*) [r-][w-]
> + */
> +
> +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;
> +
> + 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;
> +
> + 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);
> +
> + /*
> +  * No locking here is required - all that we need
> +  * is provided inside the kobject mapping code
> +  */
> +
> + 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..38057b9
> --- /dev/null
> +++ b/include/linux/devscontrol.h
> @@ -0,0 +1,26 @@
> +#ifndef __DEVS_CONTROL_H__
> +#define __DEVS_CONTROL_H__
> +struct kobj_map;
> +struct task_struct;
> +
> +/*
> + * task_[cb]dev_map - get a map from task. Both calls may return
> + * NULL, to indicate, that task doesn't belong to any group and
> + * that the global map is to be used.
> + */
> +
> +#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 732a1c2..f9a1b4f 100644
> --- a/init/Kconfig
> +++ b/init/Kconfig
> @@ -283,6 +283,19 @@ config CGROUP_DEBUG
>
>   Say N if unsure
>
> +config CGROUP_DEVS
> + bool "Devices cgroup subsystem"
> + depends on CGROUPS
> + help
> +   Controlls the access rights to devices, i.e. you may hide
> +   some of them from tasks, so that they will not be able
> +   to open them, or you may grant a read-only access to some
> +   of them.
> +
> +   See Documentation/controllers/devices.txt for details.
> +
> +   This is harmless to say N here, so do it if unsure.
> +
> 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

---

Subject: Re: [PATCH 4/4] The control group itself
Posted by Sukadev Bhattiprolu on Tue, 12 Feb 2008 07:42:17 GMT
View Forum Message <> Reply to Message

This patchset does fix the problem I was having before with null and
zero devices. Overall, it looks like pretty good.

I am still reviewing the patches. Just some nits I came across:

Pavel Emelianov [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.
|
| The top container isn't initialized, so that the char
| and block layers will use the global maps to lookup
| their devices. I did that not to export the static maps
| to the outer world.
|
| Good news is that this patch now contains more comments
| and Documentation file :)
|
| Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
|
| ---
|
| diff --git a/Documentation/controllers/devices.txt b/Documentation/controllers/devices.txt
| new file mode 100644
| index 0000000..dbd0c7a
| --- /dev/null
| +++ b/Documentation/controllers/devices.txt
| @@ -0,0 +1,61 @@
| +
| + Devices visibility controller
| +
| +This controller allows to tune the devices accessibility by tasks,
| +i.e. grant full access for /dev/null, /dev/zero etc, grant read-only
| +access to IDE devices and completely hide SCSI disks.
| +
| +Tasks still can call mknod to create device files, regardless of
| +whether the particular device is visible or accessible, but they
| +may not be able to open it later.
| +
| +This one hides under CONFIG_CGROUP_DEVS option.
| +
| +
| +Configuring
| +
| +The controller provides a single file to configure itself -- the
| +devices.permissions one. To change the accessibility level for some
| +device write the following string into it:
| +
| +[cb] <major>:(<minor>|*) [r-][w-]
| + ^   ^   ^
| + | | |

| + | | +--- access rights (1)
| + | |
| + | +-- device major and minor numbers (2)
| + |
| + +-- device type (character / block)
| +
| +1) The access rights set to '--' remove the device from the group's
| +access list, so that it will not even be shown in this file later.
| +
| +2) Setting the minor to '*' grants access to all the minors for
| +particular major.
| +
| +When reading from it, one may see something like
| +
| + c 1:5 rw
| + b 8:* r-
| +
| +Security issues, concerning who may grant access to what are governed
| +at the cgroup infrastructure level.
| +
| +
| +Examples:
| +
| +1. Grand full access to /dev/null

Grant.

| + # echo c 1:3 rw > /cgroups/<id>/devices.permissions
| +
| +2. Grant the read-only access to /dev/sda and partitions
| + # echo b 8:* r- > ...

This grants access to all scsi disks, sda..sdp and not just 'sda' right ?

| +
| +3. Change the /dev/null access to write-only
| + # echo c 1:3 -w > ...
| +
| +4. Revoke access to /dev/sda
| + # echo b 8:* -- > ...
| +
| +
| + Written by Pavel Emelyanov <xemul@openvz.org>
| +
| diff --git a/fs/Makefile b/fs/Makefile
| index 7996220..5ad03be 100644
| --- a/fs/Makefile
| +++ b/fs/Makefile

```
| @@ -64,6 +64,8 @@ obj-y    += devpts/
|
|  obj-$(CONFIG_PROFILING)  += dcookies.o
|  obj-$(CONFIG_DLM)  += dlm/
| +
| +obj-$(CONFIG_CGROUP_DEVS) += devscontrol.o
|
|  # Do not add any filesystems before this line
|  obj-$(CONFIG_REISERFS_FS) += reiserfs/
| diff --git a/fs/devscontrol.c b/fs/devscontrol.c
| new file mode 100644
| index 0000000..48c5f69
| --- /dev/null
| +++ b/fs/devscontrol.c
| @@ -0,0 +1,314 @@
| +/*
| + * devscontrol.c - Device Controller
| + *
| + * Copyright 2007 OpenVZ SWsoft Inc
| + * Author: Pavel Emelyanov <xemul at openvz dot 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 {
| + /*
| +  * The subsys state to build into cgrous infrastructure
| +  */
```

... into cgroups

```
| + struct cgroup_subsys_state css;
```

| +
| + /*
| + * The maps of character and block devices. They provide a
| + * map from dev_t-s to struct cdev/gendisk. See fs/char_dev.c
| + * and block/genhd.c to find out how the ->open() callbacks
| + * work when opening a device.
| + *
| + * Each group will have its onw maps, and at the open()

own maps

| + * time code will lookup in this map to get the device
| + * and permissions by its dev_t.
| + */
| + 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);
| +}

'devs' as prefix/suffix does not look very clear.

How about css_to_devs_cg() ? Similarly below for dev_cg_create(),
dev_cg_destroy() ?

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

---

Subject: Re: [PATCH 4/4] The control group itself
Posted by Pavel Emelianov on Tue, 12 Feb 2008 07:51:24 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> This patchset does fix the problem I was having before with null and
> zero devices. Overall, it looks like pretty good.
>
> I am still reviewing the patches. Just some nits I came across:
>
>
> Pavel Emelianov [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.
> |
> | The top container isn't initialized, so that the char
> | and block layers will use the global maps to lookup
> | their devices. I did that not to export the static maps
> | to the outer world.
> |
> | Good news is that this patch now contains more comments
> | and Documentation file :)
> |
> | Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
> |
> | ---
> |
> | diff --git a/Documentation/controllers/devices.txt b/Documentation/controllers/devices.txt
> | new file mode 100644
> | index 0000000..dbd0c7a
> | --- /dev/null
> | +++ b/Documentation/controllers/devices.txt
> | @@ -0,0 +1,61 @@
> | +
> | + Devices visibility controller
> | +
> | +This controller allows to tune the devices accessibility by tasks,
> | +i.e. grant full access for /dev/null, /dev/zero etc, grant read-only
> | +access to IDE devices and completely hide SCSI disks.
> | +
> | +Tasks still can call mknod to create device files, regardless of
> | +whether the particular device is visible or accessible, but they
> | +may not be able to open it later.
> | +
> | +This one hides under CONFIG_CGROUP_DEVS option.
> | +
> | +
> | +Configuring
> | +
> | +The controller provides a single file to configure itself -- the
> | +devices.permissions one. To change the accessibility level for some
> | +device write the following string into it:
> | +
> | +[cb] <major>:(<minor>|*) [r-][w-]
> | + ^   ^   ^
> | + | | |
> | + | | +--- access rights (1)
> | + | |
> | + | +-- device major and minor numbers (2)
> | + |

> | + +-- device type (character / block)
> | +
> | +1) The access rights set to '--' remove the device from the group's
> | +access list, so that it will not even be shown in this file later.
> | +
> | +2) Setting the minor to '*' grants access to all the minors for
> | +particular major.
> | +
> | +When reading from it, one may see something like
> | +
> | + c 1:5 rw
> | + b 8:* r-
> | +
> | +Security issues, concerning who may grant access to what are governed
> | +at the cgroup infrastructure level.
> | +
> | +
> | +Examples:
> | +
> | +1. Grand full access to /dev/null
>
> Grant.

:)

> | + # echo c 1:3 rw > /cgroups/<id>/devices.permissions
> | +
> | +2. Grant the read-only access to /dev/sda and partitions
> | + # echo b 8:* r- > ...
>
> This grants access to all scsi disks, sda..sdp and not just 'sda' right ?

Well, yes. I'll fix the comment like ;Grant the RO access to scsi disks.

> | +
> | +3. Change the /dev/null access to write-only
> | + # echo c 1:3 -w > ...
> | +
> | +4. Revoke access to /dev/sda
> | + # echo b 8:* -- > ...
> | +
> | +
> | + Written by Pavel Emelyanov <xemul@openvz.org>
> | +
> | diff --git a/fs/Makefile b/fs/Makefile
> | index 7996220..5ad03be 100644
> | --- a/fs/Makefile
> | +++ b/fs/Makefile

> | @@ -64,6 +64,8 @@ obj-y   += devpts/
> |
> | obj-$(CONFIG_PROFILING) += dcookies.o
> | obj-$(CONFIG_DLM) += dlm/
> | +
> | +obj-$(CONFIG_CGROUP_DEVS) += devscontrol.o
> |
> | # Do not add any filesystems before this line
> | obj-$(CONFIG_REISERFS_FS) += reiserfs/
> | diff --git a/fs/devscontrol.c b/fs/devscontrol.c
> | new file mode 100644
> | index 0000000..48c5f69
> | --- /dev/null
> | +++ b/fs/devscontrol.c
> | @@ -0,0 +1,314 @@
> | +/*
> | + * devscontrol.c - Device Controller
> | + *
> | + * Copyright 2007 OpenVZ SWsoft Inc
> | + * Author: Pavel Emelyanov <xemul at openvz dot 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 {
> | + /*
> | + * The subsys state to build into cgrous infrastructure
> | + */
>
> ... into cgroups
>
> | + struct cgroup_subsys_state css;

> | +
> | + /*
> | + * The maps of character and block devices. They provide a
> | + * map from dev_t-s to struct cdev/gendisk. See fs/char_dev.c
> | + * and block/genhd.c to find out how the ->open() callbacks
> | + * work when opening a device.
> | + *
> | + * Each group will have its onw maps, and at the open()
>
> own maps
>
> | + * time code will lookup in this map to get the device
> | + * and permissions by its dev_t.
> | + */
> | + 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);
> | +}
>
> 'devs' as prefix/suffix does not look very clear.
>
> How about css_to_devs_cg() ? Similarly below for dev_cg_create(),
> dev_cg_destroy() ?

These names are internal to devscontrol.c, so I'd like to keep them
as short as possible.

Thanks,
Pavel

Subject: Re: [PATCH 4/4] The control group itself
Posted by Pavel Emelianov on Tue, 12 Feb 2008 10:28:02 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> 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.
>>
>> The top container isn't initialized, so that the char
>> and block layers will use the global maps to lookup
>> their devices. I did that not to export the static maps
>> to the outer world.
>>
>> Good news is that this patch now contains more comments
>> and Documentation file :)
>
> Seems to work as advertised :) I can't reproduce Suka's null/zero
> bug.
>
> You're relying fully on uid-0 to stop writes into the
> devices.permissions files. Would you mind adding a check for
> CAP_SYS_ADMIN (or CAP_NS_OVERRIDE+CAP_MKNOD)? Or were you really
> counting on using the filesystem visibility cgroup to stop a container
> from making changes to its device access whitelist?

Yup. I strongly believe that a controller itself should not bring
any security policy of its own, but the infrastructure should
take care of this.

However, I'm open to change my mind if I see good explanation of
why it is wrong.

> thanks,
> -serge
>
>> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>>
>> ---
>>
>> diff --git a/Documentation/controllers/devices.txt b/Documentation/controllers/devices.txt
>> new file mode 100644
>> index 0000000..dbd0c7a
>> --- /dev/null
>> +++ b/Documentation/controllers/devices.txt
>> @@ -0,0 +1,61 @@
>> +
>> + Devices visibility controller
>> +
>> +This controller allows to tune the devices accessibility by tasks,
>> +i.e. grant full access for /dev/null, /dev/zero etc, grant read-only
>> +access to IDE devices and completely hide SCSI disks.
>> +
>> +Tasks still can call mknod to create device files, regardless of

>> +whether the particular device is visible or accessible, but they
>> +may not be able to open it later.
>> +
>> +This one hides under CONFIG_CGROUP_DEVS option.
>> +
>> +
>> +Configuring
>> +
>> +The controller provides a single file to configure itself -- the
>> +devices.permissions one. To change the accessibility level for some
>> +device write the following string into it:
>> +
>> +[cb] <major>:(<minor>|*) [r-][w-]
>> + ^   ^   ^
>> + | | |
>> + | | +--- access rights (1)
>> + | |
>> + | +-- device major and minor numbers (2)
>> + |
>> + +-- device type (character / block)
>> +
>> +1) The access rights set to '--' remove the device from the group's
>> +access list, so that it will not even be shown in this file later.
>> +
>> +2) Setting the minor to '*' grants access to all the minors for
>> +particular major.
>> +
>> +When reading from it, one may see something like
>> +
>> + c 1:5 rw
>> + b 8:* r-
>> +
>> +Security issues, concerning who may grant access to what are governed
>> +at the cgroup infrastructure level.
>> +
>> +
>> +Examples:
>> +
>> +1. Grand full access to /dev/null
>> + # echo c 1:3 rw > /cgroups/<id>/devices.permissions
>> +
>> +2. Grant the read-only access to /dev/sda and partitions
>> + # echo b 8:* r- > ...
>> +
>> +3. Change the /dev/null access to write-only
>> + # echo c 1:3 -w > ...
>> +
>> +4. Revoke access to /dev/sda

>> + # echo b 8:* -- > ...
>> +
>> +
>> + Written by Pavel Emelyanov <xemul@openvz.org>
>> +
>> diff --git a/fs/Makefile b/fs/Makefile
>> index 7996220..5ad03be 100644
>> --- a/fs/Makefile
>> +++ b/fs/Makefile
>> @@ -64,6 +64,8 @@ obj-y   += devpts/
>>
>> obj-$(CONFIG_PROFILING) += dcookies.o
>> obj-$(CONFIG_DLM) += dlm/
>> +
>> +obj-$(CONFIG_CGROUP_DEVS) += devscontrol.o
>>
>> # Do not add any filesystems before this line
>> obj-$(CONFIG_REISERFS_FS) += reiserfs/
>> diff --git a/fs/devscontrol.c b/fs/devscontrol.c
>> new file mode 100644
>> index 0000000..48c5f69
>> --- /dev/null
>> +++ b/fs/devscontrol.c
>> @@ -0,0 +1,314 @@
>> +/*
>> + * devscontrol.c - Device Controller
>> + *
>> + * Copyright 2007 OpenVZ SWsoft Inc
>> + * Author: Pavel Emelyanov <xemul at openvz dot 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 {
>> + /*
>> +  * The subsys state to build into cgrous infrastructure
>> +  */
>> + struct cgroup_subsys_state css;
>> +
>> + /*
>> +  * The maps of character and block devices. They provide a
>> +  * map from dev_t-s to struct cdev/gendisk. See fs/char_dev.c
>> +  * and block/genhd.c to find out how the ->open() callbacks
>> +  * work when opening a device.
>> +  *
>> +  * Each group will have its onw maps, and at the open()
>> +  * time code will lookup in this map to get the device
>> +  * and permissions by its dev_t.
>> +  */
>> + 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);
>> +}
>> +
>> +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);
>> +}
>> +
>> +/*
>> + * The devices.permissions file read/write functionality
>> + *
>> + * The following two routines parse and print the strings like
```

```
>> + * [cb] <major>:(<minor>|*) [r-][w-]
>> + */
>> +
>> +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;
>> +
>> + 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;
>> +
>> + 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);
>> +
>> + /*
>> +  * No locking here is required - all that we need
>> +  * is provided inside the kobject mapping code
```

```
>> +  */
>> +
>> + 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..38057b9
>> --- /dev/null
>> +++ b/include/linux/devscontrol.h
>> @@ -0,0 +1,26 @@
>> +#ifndef __DEVS_CONTROL_H__
>> +#define __DEVS_CONTROL_H__
>> +struct kobj_map;
>> +struct task_struct;
>> +
>> +/*
>> + * task_[cb]dev_map - get a map from task. Both calls may return
>> + * NULL, to indicate, that task doesn't belong to any group and
>> + * that the global map is to be used.
>> + */
>> +
>> +#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 732a1c2..f9a1b4f 100644
>> --- a/init/Kconfig
>> +++ b/init/Kconfig
>> @@ -283,6 +283,19 @@ config CGROUP_DEBUG
>>
>>   Say N if unsure
>>
>> +config CGROUP_DEVS
>> + bool "Devices cgroup subsystem"
>> + depends on CGROUPS
>> + help
>> +   Controlls the access rights to devices, i.e. you may hide
>> +   some of them from tasks, so that they will not be able
>> +   to open them, or you may grant a read-only access to some
>> +   of them.
>> +
>> +   See Documentation/controllers/devices.txt for details.
>> +
>> +   This is harmless to say N here, so do it if unsure.
>> +
>> 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

---

Subject: Re: [PATCH 4/4] The control group itself
Posted by serue on Tue, 12 Feb 2008 17:21:34 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> Serge E. Hallyn wrote:
> > 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.
> >>
> >> The top container isn't initialized, so that the char
> >> and block layers will use the global maps to lookup
> >> their devices. I did that not to export the static maps
> >> to the outer world.

> >>
> >> Good news is that this patch now contains more comments
> >> and Documentation file :)
> >
> > Seems to work as advertised :) I can't reproduce Suka's null/zero
> > bug.
> >
> > You're relying fully on uid-0 to stop writes into the
> > devices.permissions files. Would you mind adding a check for
> > CAP_SYS_ADMIN (or CAP_NS_OVERRIDE+CAP_MKNOD)? Or were you really
> > counting on using the filesystem visibility cgroup to stop a container
> > from making changes to its device access whitelist?
>
> Yup. I strongly believe that a controller itself should not bring
> any security policy of its own, but the infrastructure should
> take care of this.

That would be ok if the controller gave the infrastructure some way of
knowing what sort of thing the controller does. I.e. I wouldn't mind
having the cgroup infrastructe check for capabilities, but I suspect
some cgroups will really be best represented by different capabilities.

Paul (actually both Menage and Jackson :) do you have an opinion on
this? Are there sites which eg do 'chown -R some_user_id /cgroup/cpusets/'
to have some non-root user be able to dole out cpusets? Is there any
way it would be ok to have cgroup_file_write() check for CAP_SYS_ADMIN?

> However, I'm open to change my mind if I see good explanation of
> why it is wrong.

Well the thing is that it currently leaves no way to keep root in
another namespace from manipulating it. I don't think we can even use
SELinux unless we're willing to prevent containers from having write
access to everything under the cgroup - which is only ok depending on
what is composed with the devices cgroup.

We have the same kind of problem with the
/proc/sys/filesystems/<fs>/fs_safe flag. There are a few possibilities,
and your fs visibility cgroup (plus splitting /proc/sys/filesystems into
its own fs) is one. More complicated things pop into my head, but I
think until we get more comfortable with all this the simplest way is
the best.

But really imo the simplest way is to have a capable(CAP_SYS_ADMIN)
check inside your _write function :)

Ideally if you didn't mind I would float a patch (cousin to the userns
4-patch set) defining CAP_NS_OVERRIDE and requiring both CAP_NS_OVERRIDE

and CAP_SYS_ADMIN to access _write.

-serge

> > thanks,
> > -serge
> >
> >> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
> >>
> >> ---
> >>
> >> diff --git a/Documentation/controllers/devices.txt b/Documentation/controllers/devices.txt
> >> new file mode 100644
> >> index 0000000..dbd0c7a
> >> --- /dev/null
> >> +++ b/Documentation/controllers/devices.txt
> >> @@ -0,0 +1,61 @@
> >> +
> >> + Devices visibility controller
> >> +
> >> +This controller allows to tune the devices accessibility by tasks,
> >> +i.e. grant full access for /dev/null, /dev/zero etc, grant read-only
> >> +access to IDE devices and completely hide SCSI disks.
> >> +
> >> +Tasks still can call mknod to create device files, regardless of
> >> +whether the particular device is visible or accessible, but they
> >> +may not be able to open it later.
> >> +
> >> +This one hides under CONFIG_CGROUP_DEVS option.
> >> +
> >> +
> >> +Configuring
> >> +
> >> +The controller provides a single file to configure itself -- the
> >> +devices.permissions one. To change the accessibility level for some
> >> +device write the following string into it:
> >> +
> >> +[cb] <major>:(<minor>|*) [r-][w-]
> >> + ^   ^   ^
> >> + | | |
> >> + | | +--- access rights (1)
> >> + | |
> >> + | +-- device major and minor numbers (2)
> >> + |
> >> + +-- device type (character / block)
> >> +
> >> +1) The access rights set to '--' remove the device from the group's
> >> +access list, so that it will not even be shown in this file later.

> >> +
> >> +2) Setting the minor to '*' grants access to all the minors for
> >> +particular major.
> >> +
> >> +When reading from it, one may see something like
> >> +
> >> + c 1:5 rw
> >> + b 8:* r-
> >> +
> >> +Security issues, concerning who may grant access to what are governed
> >> +at the cgroup infrastructure level.
> >> +
> >> +
> >> +Examples:
> >> +
> >> +1. Grand full access to /dev/null
> >> + # echo c 1:3 rw > /cgroups/<id>/devices.permissions
> >> +
> >> +2. Grant the read-only access to /dev/sda and partitions
> >> + # echo b 8:* r- > ...
> >> +
> >> +3. Change the /dev/null access to write-only
> >> + # echo c 1:3 -w > ...
> >> +
> >> +4. Revoke access to /dev/sda
> >> + # echo b 8:* -- > ...
> >> +
> >> +
> >> + Written by Pavel Emelyanov <xemul@openvz.org>
> >> +
> >> diff --git a/fs/Makefile b/fs/Makefile
> >> index 7996220..5ad03be 100644
> >> --- a/fs/Makefile
> >> +++ b/fs/Makefile
> >> @@ -64,6 +64,8 @@ obj-y   += devpts/
> >>
> >> obj-$(CONFIG_PROFILING) += dcookies.o
> >> obj-$(CONFIG_DLM) += dlm/
> >> +
> >> +obj-$(CONFIG_CGROUP_DEVS) += devscontrol.o
> >>
> >> # Do not add any filesystems before this line
> >> obj-$(CONFIG_REISERFS_FS) += reiserfs/
> >> diff --git a/fs/devscontrol.c b/fs/devscontrol.c
> >> new file mode 100644
> >> index 0000000..48c5f69
> >> --- /dev/null
> >> +++ b/fs/devscontrol.c

```
> >> @@ -0,0 +1,314 @@
> >> +/*
> >> + * devscontrol.c - Device Controller
> >> + *
> >> + * Copyright 2007 OpenVZ SWsoft Inc
> >> + * Author: Pavel Emelyanov <xemul at openvz dot 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 {
> >> + /*
> >> +  * The subsys state to build into cgrous infrastructure
> >> +  */
> >> + struct cgroup_subsys_state css;
> >> +
> >> + /*
> >> +  * The maps of character and block devices. They provide a
> >> +  * map from dev_t-s to struct cdev/gendisk. See fs/char_dev.c
> >> +  * and block/genhd.c to find out how the ->open() callbacks
> >> +  * work when opening a device.
> >> +  *
> >> +  * Each group will have its onw maps, and at the open()
> >> +  * time code will lookup in this map to get the device
> >> +  * and permissions by its dev_t.
> >> +  */
> >> + 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);
> >> +}
> >> +
> >> +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);
> >> +}
> >> +
> >> +/*
> >> + * The devices.permissions file read/write functionality
> >> + *
> >> + * The following two routines parse and print the strings like
> >> + * [cb] <major>:(<minor>|*) [r-][w-]
> >> + */
> >> +
> >> +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;
> >> +
> >> + 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;
> >> +
> >> + 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);
> >> +
> >> + /*
> >> +  * No locking here is required - all that we need
> >> +  * is provided inside the kobject mapping code
> >> +  */
> >> +
> >> + 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..38057b9
> >> --- /dev/null
> >> +++ b/include/linux/devscontrol.h
> >> @@ -0,0 +1,26 @@
> >> +#ifndef __DEVS_CONTROL_H__
```

> >> +#define __DEVS_CONTROL_H__
> >> +struct kobj_map;
> >> +struct task_struct;
> >> +
> >> +/*
> >> + * task_[cb]dev_map - get a map from task. Both calls may return
> >> + * NULL, to indicate, that task doesn't belong to any group and
> >> + * that the global map is to be used.
> >> + */
> >> +
> >> +#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 732a1c2..f9a1b4f 100644
> >> --- a/init/Kconfig
> >> +++ b/init/Kconfig
> >> @@ -283,6 +283,19 @@ config CGROUP_DEBUG
> >>
> >>   Say N if unsure
> >>
> >> +config CGROUP_DEVS
> >> + bool "Devices cgroup subsystem"
> >> + depends on CGROUPS
> >> + help
> >> +   Controlls the access rights to devices, i.e. you may hide
> >> +   some of them from tasks, so that they will not be able
> >> +   to open them, or you may grant a read-only access to some
> >> +   of them.
> >> +
> >> +   See Documentation/controllers/devices.txt for details.
> >> +
> >> +   This is harmless to say N here, so do it if unsure.
> >> +
> >> 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

---

Subject: Re: [PATCH 4/4] The control group itself
Posted by Paul Menage on Wed, 13 Feb 2008 02:17:14 GMT
View Forum Message <> Reply to Message

On Feb 12, 2008 9:21 AM, Serge E. Hallyn <serue@us.ibm.com> wrote:
>
> Paul (actually both Menage and Jackson :) do you have an opinion on
> this? Are there sites which eg do 'chown -R some_user_id /cgroup/cpusets/'
> to have some non-root user be able to dole out cpusets?

We (Google) currently only do that for the tasks file, to allow users
to move their processes between cpusets/groups that they own.

> Is there any
> way it would be ok to have cgroup_file_write() check for CAP_SYS_ADMIN?

Sure, we could have flags in the subsys object for this kind of thing
that let particular subsystems request this.

Paul

_____

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

---

Subject: Re: [PATCH 4/4] The control group itself
Posted by Paul Jackson on Wed, 13 Feb 2008 02:32:15 GMT
View Forum Message <> Reply to Message

Serge wrote:
> Paul (actually both Menage and Jackson :) do you have an opinion on
> this? Are there sites which eg do 'chown -R some_user_id /cgroup/cpusets/'
> to have some non-root user be able to dole out cpusets? Is there any
> way it would be ok to have cgroup_file_write() check for CAP_SYS_ADMIN?

I don't know what my users actually do here ... I'm a couple layers
removed from that reality. But certainly I've recommended that they
sometimes do things like having the batch scheduler chown the files

of each jobs cpuset to the uid of the user running that job, so that
the job can manipulate its own cpuset allocate resources in finer
detail.

One of the more elaborate ways of doing this nests a pair of cpusets,
with the parent owned by the batch scheduler confining the child
owned by the individual job. The job can actually do things like
write its own cpus and mems files, but is confined by the parent
cpuset to only specify cpus and mems assigned to that job.

As to how this affects your question ... I'm not sure. However I
suspect that an added requirement for CAP_SYS_ADMIN would cause
breakage and not be a good idea.

--

I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.940.382.4214

_____

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

---

Subject: Re: [PATCH 4/4] The control group itself
Posted by Paul Jackson on Wed, 13 Feb 2008 02:42:15 GMT
View Forum Message <> Reply to Message

Paul M wrote:
> > Is there any
> > way it would be ok to have cgroup_file_write() check for CAP_SYS_ADMIN?
>
> Sure, we could have flags in the subsys object for this kind of thing
> that let particular subsystems request this.

That sounds like it would work fine ... the cpuset subsystem would
just not request this.

--

I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.940.382.4214

_____

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

---

Subject: Re: [PATCH 4/4] The control group itself
Posted by serue on Thu, 14 Feb 2008 17:17:18 GMT
View Forum Message <> Reply to Message

Quoting Paul Jackson (pj@sgi.com):
> Paul M wrote:
> > > Is there any
> > > way it would be ok to have cgroup_file_write() check for CAP_SYS_ADMIN?
> > >
> > Sure, we could have flags in the subsys object for this kind of thing
> > that let particular subsystems request this.
>
> That sounds like it would work fine ... the cpuset subsystem would
> just not request this.

Cool I'll keep it in mind then - though really right now it seems like
it just amounts to slightly more complicated approach than just having
the subsystems do the check in _write...

thanks,
-serge

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

Subject: Re: [PATCH 4/4] The control group itself
Posted by serue on Thu, 14 Feb 2008 17:18:57 GMT
View Forum Message <> Reply to Message

Quoting Paul Jackson (pj@sgi.com):
> Serge wrote:
> > Paul (actually both Menage and Jackson :) do you have an opinion on
> > this? Are there sites which eg do 'chown -R some_user_id /cgroup/cpusets/'
> > to have some non-root user be able to dole out cpusets? Is there any
> > way it would be ok to have cgroup_file_write() check for CAP_SYS_ADMIN?
>
> I don't know what my users actually do here ... I'm a couple layers
> removed from that reality. But certainly I've recommended that they
> sometimes do things like having the batch scheduler chown the files
> of each jobs cpuset to the uid of the user running that job, so that
> the job can manipulate its own cpuset allocate resources in finer
> detail.
>
> One of the more elaborate ways of doing this nests a pair of cpusets,
> with the parent owned by the batch scheduler confining the child
> owned by the individual job. The job can actually do things like

> write its own cpus and mems files, but is confined by the parent
> cpuset to only specify cpus and mems assigned to that job.
>
> As to how this affects your question ... I'm not sure. However I
> suspect that an added requirement for CAP_SYS_ADMIN would cause
> breakage and not be a good idea.

Yeah, I guess a more general mechanism to couple a user namespace's
connection to a mount is the right way to go. If we can just specify
that root in this namespace is not root in that namespace (or any other
userid we've chowned the files to), we've got what we need.

thanks,
-serge

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

---

Subject: Re: [PATCH 4/4] The control group itself
Posted by Paul Menage on Thu, 21 Feb 2008 20:47:07 GMT
View Forum Message <> Reply to Message

On Thu, Feb 7, 2008 at 5:01 AM, Pavel Emelyanov <xemul@openvz.org> wrote:
> +
> +[cb] <major>:(<minor>|*) [r-][w-]
> + ^ ^ ^
> + | | |
> + | | +--- access rights (1)
> + | |
> + | +-- device major and minor numbers (2)
> + |
> + +-- device type (character / block)
> ...
> +When reading from it, one may see something like
> +
> + c 1:5 rw
> + b 8:* r-
> +

In the interest of avoiding proliferating cgroup control file formats,
I'm wondering if we can abstract out the general form of the data
being presented here and maybe simplify it in such a way that we can
hopefully reuse the format for other control files in the future?

For example, one way to represent this would be a map from device
strings such c:1:5 to permission strings such as rw. Or maybe

numerical device ids to numerical permission values.

The alternative might be to accept that there are two kinds of control
files - those which are likely to be programmatically read (e.g.
resource usage values), and those that are likely to be
programmatically written but only actually read by humans for
debugging purposes (like this one) and make it clear up-front when a
control file is added which type they're considered to be. We could
then ignore the API consistency requirements for the
debugging-readable files.

On a separate note, can you document the recommended way to completely
overwrite the set of device permissions for a cgroup? Does this
involves writing a "--" permission for every device that you don't
want in the cgroup?

Paul

Subject: Re: [PATCH 4/4] The control group itself
Posted by Pavel Emelianov on Fri, 22 Feb 2008 08:12:45 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On Thu, Feb 7, 2008 at 5:01 AM, Pavel Emelyanov <xemul@openvz.org> wrote:
>> +
>> +[cb] <major>:(<minor>|*) [r-][w-]
>> + ^ ^ ^
>> + | | |
>> + | | +--- access rights (1)
>> + | |
>> + | +-- device major and minor numbers (2)
>> + |
>> + +-- device type (character / block)
>> ...
>> +When reading from it, one may see something like
>> +
>> + c 1:5 rw
>> + b 8:* r-
>> +
>
> In the interest of avoiding proliferating cgroup control file formats,
> I'm wondering if we can abstract out the general form of the data
> being presented here and maybe simplify it in such a way that we can

> hopefully reuse the format for other control files in the future?
>
> For example, one way to represent this would be a map from device
> strings such c:1:5 to permission strings such as rw. Or maybe
> numerical device ids to numerical permission values.

You mean smth like <some_device_id><space><some_permissions_string>?

Well, I don't mind, but AFAIK the <major>:<minor> form is very common
for specifying the device. So I agree with the 'c:1:5 rw' form.

> The alternative might be to accept that there are two kinds of control
> files - those which are likely to be programmatically read (e.g.
> resource usage values), and those that are likely to be
> programmatically written but only actually read by humans for
> debugging purposes (like this one) and make it clear up-front when a
> control file is added which type they're considered to be. We could
> then ignore the API consistency requirements for the
> debugging-readable files.

Hmm, you mean make them a binary files? I thought that filesystem-based
API should be human readable and writable as much as possible...

> On a separate note, can you document the recommended way to completely
> overwrite the set of device permissions for a cgroup? Does this

There's not way to flush all the permissions in this implementation, but
I though about one. Maybe 'echo 0 > devices.permissions' would be good?

> involves writing a "--" permission for every device that you don't
> want in the cgroup?

Currently - yes.

> Paul
>

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

---

Subject: Re: [PATCH 4/4] The control group itself
Posted by Paul Menage on Sat, 23 Feb 2008 23:12:59 GMT
View Forum Message <> Reply to Message

On Fri, Feb 22, 2008 at 12:12 AM, Pavel Emelyanov <xemul@openvz.org> wrote:

>
> Hmm, you mean make them a binary files?

No, not by default. But I'm working on a plan to have an optional
binary API to cgroups, that would allow multiple control files to be
read in a binary format with a single system call. The existing API
would still be available as well, of course. The idea would be that
monitoring programs that frequently read lots of values from a single
cgroup (or even multiple cgroups) would be able to do so more cheaply
than by making multiple different reads on different files.

In order for this to work, CGroups needs to know the data type of a
given control file - so this would only be available for the control
files that use typed cgroup output methods rather than the raw file
output interface.

> I thought that filesystem-based
> API should be human readable and writable as much as possible...
>

Yes, but even without a binary API it makes sense for values that are
likely to be parsed by programs be in a consistent format.

But after thinking more about this, I think that the devices
permission control file output doesn't really fall under this category
- from a programmatic point of view, I suspect it's write-only, and
only humans will be reading the output, for debugging.

Paul

Subject: Re: [PATCH 4/4] The control group itself
Posted by Pavel Emelianov on Tue, 26 Feb 2008 07:54:09 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On Fri, Feb 22, 2008 at 12:12 AM, Pavel Emelyanov <xemul@openvz.org> wrote:
>> Hmm, you mean make them a binary files?
>
> No, not by default. But I'm working on a plan to have an optional
> binary API to cgroups, that would allow multiple control files to be
> read in a binary format with a single system call. The existing API
> would still be available as well, of course. The idea would be that
> monitoring programs that frequently read lots of values from a single

> cgroup (or even multiple cgroups) would be able to do so more cheaply
> than by making multiple different reads on different files.
>
> In order for this to work, CGroups needs to know the data type of a
> given control file - so this would only be available for the control
> files that use typed cgroup output methods rather than the raw file
> output interface.

Sounds reasonable.

>> I thought that filesystem-based
>> API should be human readable and writable as much as possible...
>>
>
> Yes, but even without a binary API it makes sense for values that are
> likely to be parsed by programs be in a consistent format.
>
> But after thinking more about this, I think that the devices
> permission control file output doesn't really fall under this category
> - from a programmatic point of view, I suspect it's write-only, and
> only humans will be reading the output, for debugging.

Yup. So, if you're fine with the proposed API, I think I will prepare
this set and send it to Andrew this week.

> Paul
>

Thanks,
Pavel

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