Subject: [PATCH 0/4] Devices accessibility control group (v2)
Posted by Pavel Emelianov on Tue, 08 Jan 2008 09:02:50 GMT
View Forum Message <> Reply to Message

The first version was posted long ago
(http://openvz.org/pipermail/devel/2007-September/007647.html)
and since then there are many (good I hope) changes:

* 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.

That is - the main features I wished to implement right
after the v1 was sent. Some minor changes are:

* I merged the devices.char and devices.block files into
  one - devices.permissions;
* As the result of the change above - the strings passed
  to this file has changed. Now they are
        [bc] <major>:{<minor>|*} [r-][w-]
  E.g. b 5:2 r- will grant the read permissions to the
  block 5:2 device and c 3:* -w will grant the write-only
  access to all the character devices with the major 5.

However, there are some things to be done:

* Make the /proc/devices show relevant info depending on
  who is reading it. This seems to be easy to do, since
  I already have the support to dump similar info into the
  devices.permissions file, but I haven't tried to use
  this in /proc/devices yet;
* Add the support for devices ranges. I.e. someone might
  wish to tell smth like b 5:[0-10] r- to this subsystem.
  Currently this is not supported and I'm afraid that if we
  start support minor ranges we'll have smth similar to
  VMA-s or FLOCK-s ranges management in one more place in the
  kernel.
* One more question is - are there any other permissions to
  work with? E.g. in OpenVZ we have a separate bit for
  quota management, maybe we can invent some more...

Currently I didn'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 the v3 and further.

The set is prepared against the 2.6.24-rc5-mm1.

All this is minimally tested and seems to work. Hope to hear
you comments, wishes and patches soon :)

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: [PATCH 1/4] Some changes in the kobject mapper
Posted by Pavel Emelianov on Tue, 08 Jan 2008 09:06:46 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
Posted by Pavel Emelianov on Tue, 08 Jan 2008 09:12:51 GMT
View Forum Message <> Reply to Message

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.

To address this issue, I store the last_perm and last(_map)
fields on the struct cdev (and protect them with the cdev_lock)
and force the re-lookup in the kobj mappings if needed.

I know, this might be slow, but I have two points for it:
1. The re-lookup happens on open() only which is not
   a fast-path. Besides, this is so for block layer and
   nobody complains;
2. On a well-isolated setup, when each container has its
   own filesystem this is no longer a problem - each
   cgroup will cache the cdev on its inode and work good.

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

```
diff --git a/fs/char_dev.c b/fs/char_dev.c
index c3bfa76..2b821ef 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 +364,25 @@ int chrdev_open(struct inode * inode, struct file * filp)
  struct cdev *p;
  struct cdev *new = NULL;
  int ret = 0;
+ struct kobj_map *map;
+ mode_t mode;
+
+ map = task_cdev_map(current);
+ if (map == NULL)
+  map = cdev_map;

  spin_lock(&cdev_lock);
  p = inode->i_cdev;
- if (!p) {
+ if (!p || p->last != map) {
   struct kobject *kobj;
   int idx;
+
   spin_unlock(&cdev_lock);
-  kobj = kobj_lookup(cdev_map, inode->i_rdev, &idx);
+  kobj = kobj_lookup(map, inode->i_rdev, &mode, &idx);
   if (!kobj)
    return -ENXIO;
   new = container_of(kobj, struct cdev, kobj);
+  BUG_ON(p != NULL && p != new);
   spin_lock(&cdev_lock);
   p = inode->i_cdev;
   if (!p) {
@@ -382,12 +392,24 @@ int chrdev_open(struct inode * inode, struct file * filp)
    new = NULL;
   } else if (!cdev_get(p))
    ret = -ENXIO;
+  else {
+   p->last = map;
+   p->last_mode = mode;
+  }
  } else if (!cdev_get(p))
   ret = -ENXIO;
```

```
+ else
+  mode = p->last_mode;
  spin_unlock(&cdev_lock);
  cdev_put(new);
  if (ret)
   return ret;
+
+ if ((filp->f_mode & mode) != filp->f_mode) {
+  cdev_put(p);
+  return -EACCES;
+ }
+
  filp->f_op = fops_get(p->ops);
  if (!filp->f_op) {
   cdev_put(p);
@@ -461,6 +483,64 @@ 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
+static inline void cdev_map_reset(struct kobj_map *map, struct cdev *c)
+{
+ spin_lock(&cdev_lock);
+ if (c->last == map)
+  c->last = NULL;
+ spin_unlock(&cdev_lock);
+}
+
+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;
+ }
+
+ cdev_map_reset(map, c);
+ 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_map_reset(map, c);
+
+ 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);
@@ -542,9 +622,19 @@ 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..d72a2a1 100644
--- a/include/linux/cdev.h
+++ b/include/linux/cdev.h
@@ -9,6 +9,7 @@
 struct file_operations;
 struct inode;
 struct module;
+struct kobj_map;

 struct cdev {
  struct kobject kobj;
@@ -17,6 +18,8 @@ struct cdev {
  struct list_head list;
  dev_t dev;
  unsigned int count;
+ struct kobj_map *last;
+ mode_t last_mode;
 };

 void cdev_init(struct cdev *, const struct file_operations *);
@@ -33,5 +36,11 @@ 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 Tue, 08 Jan 2008 09:15:09 GMT

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 dc710a0..b2d7b52 100644
--- a/include/linux/genhd.h
+++ b/include/linux/genhd.h
@@ -239,7 +239,15 @@ 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);
```

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

Subject: [PATCH 4/4] The control group itself
Posted by Pavel Emelianov on Tue, 08 Jan 2008 09:18:03 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. E.g.

cat /cont/devices/0/devices.permissions
c 1:* rw
b 8:1 rw

The top container isn't initialized, so that the char
and block layers will use the global maps to lookup
their devices.

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

---

diff --git a/fs/Makefile b/fs/Makefile
index 82b6ae1..a085706 100644
--- a/fs/Makefile
+++ b/fs/Makefile
@@ -63,6 +63,8 @@ obj-y   += devpts/

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

 # Do not add any filesystems before this line
 obj-$(CONFIG_REISERFS_FS) += reiserfs/
diff --git a/fs/devscontrol.c b/fs/devscontrol.c
new file mode 100644
index 0000000..ea282f3
--- /dev/null
+++ b/fs/devscontrol.c
@@ -0,0 +1,291 @@
+/*
+ * devscontrol.c - Device Controller
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ * Author: Pavel Emelyanov <xemul at openvz.org>
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be useful,

```
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
+ * GNU General Public License for more details.
+ */
+
+#include <linux/cgroup.h>
+#include <linux/cdev.h>
+#include <linux/err.h>
+#include <linux/devscontrol.h>
+#include <linux/uaccess.h>
+#include <linux/fs.h>
+#include <linux/genhd.h>
+
+struct devs_cgroup {
+ struct cgroup_subsys_state css;
+
+ struct kobj_map *cdev_map;
+ struct kobj_map *bdev_map;
+};
+
+static inline
+struct devs_cgroup *css_to_devs(struct cgroup_subsys_state *css)
+{
+ return container_of(css, struct devs_cgroup, css);
+}
+
+static inline
+struct devs_cgroup *cgroup_to_devs(struct cgroup *cont)
+{
+ return css_to_devs(cgroup_subsys_state(cont, devs_subsys_id));
+}
+
+struct kobj_map *task_cdev_map(struct task_struct *tsk)
+{
+ struct cgroup_subsys_state *css;
+
+ css = task_subsys_state(tsk, devs_subsys_id);
+ if (css->cgroup->parent == NULL)
+  return NULL;
+ else
+  return css_to_devs(css)->cdev_map;
+}
+
+struct kobj_map *task_bdev_map(struct task_struct *tsk)
+{
+ struct cgroup_subsys_state *css;
+
+ css = task_subsys_state(tsk, devs_subsys_id);
```

```
+ if (css->cgroup->parent == NULL)
+  return NULL;
+ else
+  return css_to_devs(css)->bdev_map;
+}
+
+static struct cgroup_subsys_state *
+devs_create(struct cgroup_subsys *ss, struct cgroup *cont)
+{
+ struct devs_cgroup *devs;
+
+ devs = kzalloc(sizeof(struct devs_cgroup), GFP_KERNEL);
+ if (devs == NULL)
+  goto out;
+
+ devs->cdev_map = cdev_map_init();
+ if (devs->cdev_map == NULL)
+  goto out_free;
+
+ devs->bdev_map = bdev_map_init();
+ if (devs->bdev_map == NULL)
+  goto out_free_cdev;
+
+ return &devs->css;
+
+out_free_cdev:
+ cdev_map_fini(devs->cdev_map);
+out_free:
+ kfree(devs);
+out:
+ return ERR_PTR(-ENOMEM);
+}
+
+static void devs_destroy(struct cgroup_subsys *ss, struct cgroup *cont)
+{
+ struct devs_cgroup *devs;
+
+ devs = cgroup_to_devs(cont);
+ bdev_map_fini(devs->bdev_map);
+ cdev_map_fini(devs->cdev_map);
+ kfree(devs);
+}
+
+static int decode_perms_str(char *buf, int *chrdev, dev_t *dev,
+  int *all, mode_t *mode)
+{
+ unsigned int major, minor;
+ char *end;
```

```
+ mode_t tmp;
+
+ /* [cb] <major>:<minor> [r-][w-] */
+
+ if (buf[0] == 'c')
+  *chrdev = 1;
+ else if (buf[0] == 'b')
+  *chrdev = 0;
+ else
+  return -EINVAL;
+
+ if (buf[1] != ' ')
+  return -EINVAL;
+
+ major = simple_strtoul(buf + 2, &end, 10);
+ if (*end != ':')
+  return -EINVAL;
+
+ if (end[1] == '*') {
+  if (end[2] != ' ')
+   return -EINVAL;
+
+  *all = 1;
+  minor = 0;
+  end += 2;
+ } else {
+  minor = simple_strtoul(end + 1, &end, 10);
+  if (*end != ' ')
+   return -EINVAL;
+
+  *all = 0;
+ }
+
+ tmp = 0;
+
+ if (end[1] == 'r')
+  tmp |= FMODE_READ;
+ else if (end[1] != '-')
+  return -EINVAL;
+ if (end[2] == 'w')
+  tmp |= FMODE_WRITE;
+ else if (end[2] != '-')
+  return -EINVAL;
+
+ *dev = MKDEV(major, minor);
+ *mode = tmp;
+ return 0;
+}
```

```
+
+static int encode_perms_str(char *buf, int len, int chrdev, dev_t dev,
+  int all, mode_t mode)
+{
+ int ret;
+
+ ret = snprintf(buf, len, "%c %d:", chrdev ? 'c' : 'b', MAJOR(dev));
+ if (all)
+  ret += snprintf(buf + ret, len - ret, "*");
+ else
+  ret += snprintf(buf + ret, len - ret, "%d", MINOR(dev));
+
+ ret += snprintf(buf + ret, len - ret, " %c%c\n",
+   (mode & FMODE_READ) ? 'r' : '-',
+   (mode & FMODE_WRITE) ? 'w' : '-');
+
+ return ret + 1;
+}
+
+static ssize_t devs_write(struct cgroup *cont, struct cftype *cft,
+  struct file *f, const char __user *ubuf,
+  size_t nbytes, loff_t *pos)
+{
+ int err, all, chrdev;
+ dev_t dev;
+ char buf[64];
+ struct devs_cgroup *devs;
+ mode_t mode;
+
+ if (copy_from_user(buf, ubuf, sizeof(buf)))
+  return -EFAULT;
+
+ buf[sizeof(buf) - 1] = 0;
+ err = decode_perms_str(buf, &chrdev, &dev, &all, &mode);
+ if (err < 0)
+  return err;
+
+ devs = cgroup_to_devs(cont);
+
+ if (mode == 0) {
+  if (chrdev)
+   err = cdev_del_from_map(devs->cdev_map, dev, all);
+  else
+   err = bdev_del_from_map(devs->bdev_map, dev, all);
+
+  if (err < 0)
+   return err;
+
```

```
+  css_put(&devs->css);
+ } else {
+  if (chrdev)
+   err = cdev_add_to_map(devs->cdev_map, dev, all, mode);
+  else
+   err = bdev_add_to_map(devs->bdev_map, dev, all, mode);
+
+  if (err < 0)
+   return err;
+
+  css_get(&devs->css);
+ }
+
+ return nbytes;
+}
+
+struct devs_dump_arg {
+ char *buf;
+ int pos;
+ int chrdev;
+};
+
+static int devs_dump_one(dev_t dev, int range, mode_t mode, void *x)
+{
+ struct devs_dump_arg *arg = x;
+ char tmp[64];
+ int len;
+
+ len = encode_perms_str(tmp, sizeof(tmp), arg->chrdev, dev,
+   range != 1, mode);
+
+ if (arg->pos >= PAGE_SIZE - len)
+  return 1;
+
+ memcpy(arg->buf + arg->pos, tmp, len);
+ arg->pos += len;
+ return 0;
+}
+
+static ssize_t devs_read(struct cgroup *cont, struct cftype *cft,
+  struct file *f, char __user *ubuf, size_t nbytes, loff_t *pos)
+{
+ struct devs_dump_arg arg;
+ struct devs_cgroup *devs;
+ ssize_t ret;
+
+ arg.buf = (char *)__get_free_page(GFP_KERNEL);
+ if (arg.buf == NULL)
```

```
+  return -ENOMEM;
+
+ devs = cgroup_to_devs(cont);
+ arg.pos = 0;
+
+ arg.chrdev = 1;
+ cdev_iterate_map(devs->cdev_map, devs_dump_one, &arg);
+
+ arg.chrdev = 0;
+ bdev_iterate_map(devs->bdev_map, devs_dump_one, &arg);
+
+ ret = simple_read_from_buffer(ubuf, nbytes, pos,
+    arg.buf, arg.pos);
+
+ free_page((unsigned long)arg.buf);
+ return ret;
+}
+
+static struct cftype devs_files[] = {
+ {
+   .name = "permissions",
+   .write = devs_write,
+   .read = devs_read,
+ },
+};
+
+static int devs_populate(struct cgroup_subsys *ss, struct cgroup *cont)
+{
+ return cgroup_add_files(cont, ss,
+    devs_files, ARRAY_SIZE(devs_files));
+}
+
+struct cgroup_subsys devs_subsys = {
+ .name = "devices",
+ .subsys_id = devs_subsys_id,
+ .create = devs_create,
+ .destroy = devs_destroy,
+ .populate = devs_populate,
+};
diff --git a/include/linux/cgroup_subsys.h b/include/linux/cgroup_subsys.h
index 228235c..9c0cd2c 100644
--- a/include/linux/cgroup_subsys.h
+++ b/include/linux/cgroup_subsys.h
@@ -42,3 +42,9 @@ SUBSYS(mem_cgroup)
 #endif

 /* */
+
```

```
+#ifdef CONFIG_CGROUP_DEVS
+SUBSYS(devs)
+#endif
+
+/* */
diff --git a/include/linux/devscontrol.h b/include/linux/devscontrol.h
new file mode 100644
index 0000000..5f574e0
--- /dev/null
+++ b/include/linux/devscontrol.h
@@ -0,0 +1,20 @@
+#ifndef __DEVS_CONTROL_H__
+#define __DEVS_CONTROL_H__
+struct kobj_map;
+struct task_struct;
+
+#ifdef CONFIG_CGROUP_DEVS
+struct kobj_map *task_cdev_map(struct task_struct *);
+struct kobj_map *task_bdev_map(struct task_struct *);
+#else
+static inline kobj_map *task_cdev_map(struct task_struct *tsk)
+{
+ return NULL;
+}
+
+static inline kobj_map *task_bdev_map(struct task_struct *tsk)
+{
+ return NULL;
+}
+#endif
+#endif
diff --git a/init/Kconfig b/init/Kconfig
index b886ac9..4dd53d6 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -283,6 +283,12 @@ config CGROUP_DEBUG

    Say N if unsure

+config CGROUP_DEVS
+ bool "Devices cgroup subsystem"
+ depends on CGROUPS
+ help
+   Controlls the visibility of devices
+
 config CGROUP_NS
        bool "Namespace cgroup subsystem"
        depends on CGROUPS
```

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

Subject: Re: [PATCH 1/4] Some changes in the kobject mapper
Posted by Daniel Hokka Zakrisso on Tue, 08 Jan 2008 18:36:36 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> 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>
>
> ...
> @@ -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++) {

This should be 255, shouldn't it?

> + 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

--
Daniel Hokka Zakrisson

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

---

Subject: Re: [PATCH 1/4] Some changes in the kobject mapper
Posted by Dave Hansen on Tue, 08 Jan 2008 19:17:28 GMT
View Forum Message <> Reply to Message

On Tue, 2008-01-08 at 19:36 +0100, Daniel Hokka Zakrisson wrote:
>
> > +void kobj_map_fini(struct kobj_map *map)
> > +{

> > + int i;
> > + struct probe *p, *next;
> > +
> > + for (i = 0; i < 256; i++) {
>
> This should be 255, shouldn't it?

Neither, it should be a named constant. ;)

-- Dave

_____
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 (v2)
Posted by Sukadev Bhattiprolu on Sat, 12 Jan 2008 21:20:14 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@openvz.org] wrote:
| The first version was posted long ago
| (http://openvz.org/pipermail/devel/2007-September/007647.html)
| and since then there are many (good I hope) changes:
|
| * 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.
|
| That is - the main features I wished to implement right
| after the v1 was sent. Some minor changes are:
|
| * I merged the devices.char and devices.block files into
| one - devices.permissions;
| * As the result of the change above - the strings passed
| to this file has changed. Now they are
| [bc] <major>:{<minor>|*} [r-][w-]
| E.g. b 5:2 r- will grant the read permissions to the
| block 5:2 device and c 3:* -w will grant the write-only
| access to all the character devices with the major 5.
|
| However, there are some things to be done:

|
| * Make the /proc/devices show relevant info depending on
| who is reading it. This seems to be easy to do, since
| I already have the support to dump similar info into the
| devices.permissions file, but I haven't tried to use
| this in /proc/devices yet;
| * Add the support for devices ranges. I.e. someone might
| wish to tell smth like b 5:[0-10] r- to this subsystem.
| Currently this is not supported and I'm afraid that if we
| start support minor ranges we'll have smth similar to
| VMA-s or FLOCK-s ranges management in one more place in the
| kernel.
| * One more question is - are there any other permissions to
| work with? E.g. in OpenVZ we have a separate bit for
| quota management, maybe we can invent some more...
|
| Currently I didn'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 the v3 and further.
|
| The set is prepared against the 2.6.24-rc5-mm1.
|
| All this is minimally tested and seems to work. Hope to hear
| you comments, wishes and patches soon :)
|
| To play with it - run a standard procedure:
|
| # mount -t container none /cont/devs -o devices

This should be '-t cgroup'

| # mkdir /cont/devs/0
| # echo -n $$ > /cont/devs/0/tasks
|
| and tune device permissions.

I started playing with this and noticed that even if I try to
enable read access to device [c, 1:3] it also grants access
to device [c, 1:5].

i.e the access restrictions seem to apply to all devices with
a given major number. Is that really the intent ?

Both devices accessible here:
 # hexdump /dev/null
 # hexdump /dev/zero
 0000000 0000 0000 0000 0000 0000 0000 0000 0000

```
*
^C
```

Neither device accessible:

```
# echo $$ > /container/devs/0/tasks
# hexdump /dev/zero
hexdump: /dev/zero: No such device or address
hexdump: /dev/zero: Bad file descriptor
# hexdump /dev/null
hexdump: /dev/null: No such device or address
hexdump: /dev/null: Bad file descriptor
```

Grant read access to /dev/null, but /dev/zero is also readable

```
# echo c 1:3 r- > /container/devs/0/devices.permissions
# hexdump /dev/null
# hexdump /dev/zero
0000000 0000 0000 0000 0000 0000 0000 0000 0000
*
^C
```

Remove read access to /dev/null, but /dev/zero is also not
readable.

```
# echo c 1:3 -- > /container/devs/0/devices.permissions
# hexdump /dev/zero
hexdump: /dev/zero: No such device or address
hexdump: /dev/zero: Bad file descriptor
```

BTW, a question about cgroups: If we 'echo $$ > /container/devs/0/tasks'
is there a way to remove/undo it later (so that the process has access
as before) ?

_____
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 (v2)
Posted by Pavel Emelianov on Mon, 14 Jan 2008 07:52:59 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> Pavel Emelianov [xemul@openvz.org] wrote:
> | The first version was posted long ago
> | (http://openvz.org/pipermail/devel/2007-September/007647.html)

> | and since then there are many (good I hope) changes:
> |
> | * 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.
> |
> | That is - the main features I wished to implement right
> | after the v1 was sent. Some minor changes are:
> |
> | * I merged the devices.char and devices.block files into
> | one - devices.permissions;
> | * As the result of the change above - the strings passed
> | to this file has changed. Now they are
> | [bc] <major>:{<minor>|*} [r-][w-]
> | E.g. b 5:2 r- will grant the read permissions to the
> | block 5:2 device and c 3:* -w will grant the write-only
> | access to all the character devices with the major 5.
> |
> | However, there are some things to be done:
> |
> | * Make the /proc/devices show relevant info depending on
> | who is reading it. This seems to be easy to do, since
> | I already have the support to dump similar info into the
> | devices.permissions file, but I haven't tried to use
> | this in /proc/devices yet;
> | * Add the support for devices ranges. I.e. someone might
> | wish to tell smth like b 5:[0-10] r- to this subsystem.
> | Currently this is not supported and I'm afraid that if we
> | start support minor ranges we'll have smth similar to
> | VMA-s or FLOCK-s ranges management in one more place in the
> | kernel.
> | * One more question is - are there any other permissions to
> | work with? E.g. in OpenVZ we have a separate bit for
> | quota management, maybe we can invent some more...
> |
> | Currently I didn'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 the v3 and further.
> |
> | The set is prepared against the 2.6.24-rc5-mm1.
> |
> | All this is minimally tested and seems to work. Hope to hear

> | you comments, wishes and patches soon :)
> |
> | To play with it - run a standard procedure:
> |
> | # mount -t container none /cont/devs -o devices
>
> This should be '-t cgroup'

Right :)

Thank you for the feedback. Serge, Oren, do you have smth
to tell about this set? I planned to show it to Andrew this
week, hope he will find time to look at it :)

> | # mkdir /cont/devs/0
> | # echo -n $$ > /cont/devs/0/tasks
> |
> | and tune device permissions.
>
> I started playing with this and noticed that even if I try to
> enable read access to device [c, 1:3] it also grants access
> to device [c, 1:5].

Hm... I can't reproduce this:

/bin/echo 'c 1:3 r-' > /cnt/dev/0/devices.permissions
/bin/echo -n $$ > /cnt/dev/0/tasks
cat /cnt/dev/0/devices.permissions
c 1:3 r-
hexdump /dev/null
hexdump /dev/zero
hexdump: /dev/zero: No such device or address
hexdump: /dev/zero: Bad file descriptor

Maybe you have played with devs cgroups before getting this?
Can you show what's the contents of the devices.permissions file
in your case?

> i.e the access restrictions seem to apply to all devices with
> a given major number. Is that really the intent ?
>
> Both devices accessible here:
> # hexdump /dev/null
> # hexdump /dev/zero
> 0000000 0000 0000 0000 0000 0000 0000 0000 0000
> *
> ^C
>

> Neither device accessible:
>
> # echo $$ > /container/devs/0/tasks
> # hexdump /dev/zero
> hexdump: /dev/zero: No such device or address
> hexdump: /dev/zero: Bad file descriptor
> # hexdump /dev/null
> hexdump: /dev/null: No such device or address
> hexdump: /dev/null: Bad file descriptor
>
> Grant read access to /dev/null, but /dev/zero is also readable
>
> # echo c 1:3 r- > /container/devs/0/devices.permissions
> # hexdump /dev/null
> # hexdump /dev/zero
> 0000000 0000 0000 0000 0000 0000 0000 0000 0000
> *
> ^C
>
> Remove read access to /dev/null, but /dev/zero is also not
> readable.
>
> # echo c 1:3 -- > /container/devs/0/devices.permissions
> # hexdump /dev/zero
> hexdump: /dev/zero: No such device or address
> hexdump: /dev/zero: Bad file descriptor
>
> BTW, a question about cgroups: If we 'echo $$ > /container/devs/0/tasks'
> is there a way to remove/undo it later (so that the process has access
> as before) ?

I've always thought that it's to move the task to top cgrop, i.e.
echo $$ > /container/devs/tasks

Thanks,
Pavel

>

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

Subject: Re: [PATCH 2/4] The character devices layer changes
Posted by serue on Mon, 14 Jan 2008 17:03:33 GMT

Quoting Pavel Emelyanov (xemul@openvz.org):
> 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.
>
> To address this issue, I store the last_perm and last(_map)
> fields on the struct cdev (and protect them with the cdev_lock)
> and force the re-lookup in the kobj mappings if needed.
>
> I know, this might be slow, but I have two points for it:
> 1. The re-lookup happens on open() only which is not
>   a fast-path. Besides, this is so for block layer and
>   nobody complains;
> 2. On a well-isolated setup, when each container has its
>   own filesystem this is no longer a problem - each
>   cgroup will cache the cdev on its inode and work good.

What about simply returning -EPERM when open()ing a cdev
with ->map!=task_cdev_map(current)?

Shouldn't be a problem for ttys, since the container init
already has the tty open, right?

Otherwise, the patchset looks good to me. Want to look
through this one a little more (i think that'd be easier
with the -EPERM approach) and scrutinize patch 4, but
overall it makes sense.

If I understand right, we're taking 14k per cgroup for
kobjmaps? Do we consider that a problem?

thanks,
-serge

> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> ---
>
> diff --git a/fs/char_dev.c b/fs/char_dev.c
> index c3bfa76..2b821ef 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 +364,25 @@ int chrdev_open(struct inode * inode, struct file * filp)
>   struct cdev *p;
>   struct cdev *new = NULL;
>   int ret = 0;
> + struct kobj_map *map;
> + mode_t mode;
> +
> + map = task_cdev_map(current);
> + if (map == NULL)
> +  map = cdev_map;
>
>   spin_lock(&cdev_lock);
>   p = inode->i_cdev;
> - if (!p) {
> + if (!p || p->last != map) {
>    struct kobject *kobj;
>    int idx;
> +
>    spin_unlock(&cdev_lock);
> -  kobj = kobj_lookup(cdev_map, inode->i_rdev, &idx);
> +  kobj = kobj_lookup(map, inode->i_rdev, &mode, &idx);
>    if (!kobj)
>     return -ENXIO;
>    new = container_of(kobj, struct cdev, kobj);
> +  BUG_ON(p != NULL && p != new);
>    spin_lock(&cdev_lock);
>    p = inode->i_cdev;
>    if (!p) {
> @@ -382,12 +392,24 @@ int chrdev_open(struct inode * inode, struct file * filp)
>     new = NULL;
>    } else if (!cdev_get(p))
>     ret = -ENXIO;
> +  else {
> +   p->last = map;
> +   p->last_mode = mode;
> +  }
>   } else if (!cdev_get(p))
>    ret = -ENXIO;
> + else
```

```
> +  mode = p->last_mode;
>   spin_unlock(&cdev_lock);
>   cdev_put(new);
>   if (ret)
>    return ret;
> +
> + if ((filp->f_mode & mode) != filp->f_mode) {
> +  cdev_put(p);
> +  return -EACCES;
> + }
> +
>   filp->f_op = fops_get(p->ops);
>   if (!filp->f_op) {
>    cdev_put(p);
> @@ -461,6 +483,64 @@ 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
> +static inline void cdev_map_reset(struct kobj_map *map, struct cdev *c)
> +{
> + spin_lock(&cdev_lock);
> + if (c->last == map)
> +  c->last = NULL;
> + spin_unlock(&cdev_lock);
> +}
> +
> +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;
> + }
> +
> + cdev_map_reset(map, c);
> + 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_map_reset(map, c);
> +
> + 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);
> @@ -542,9 +622,19 @@ 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..d72a2a1 100644
> --- a/include/linux/cdev.h
> +++ b/include/linux/cdev.h
> @@ -9,6 +9,7 @@
>  struct file_operations;
>  struct inode;
>  struct module;
> +struct kobj_map;
>
>  struct cdev {
>   struct kobject kobj;
> @@ -17,6 +18,8 @@ struct cdev {
>   struct list_head list;
>   dev_t dev;
>   unsigned int count;
> + struct kobj_map *last;
> + mode_t last_mode;
>  };
>
>  void cdev_init(struct cdev *, const struct file_operations *);
> @@ -33,5 +36,11 @@ 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: Re: [PATCH 4/4] The control group itself
Posted by serue on Mon, 14 Jan 2008 17:40:56 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):

> Each new group will have its own maps for char and block
> layers. The devices access list is tuned via the
> devices.permissions file.
>
> One may read from the file to get the configured
> state. E.g.
>
> # cat /cont/devices/0/devices.permissions
> c 1:* rw
> b 8:1 rw
>
> The top container isn't initialized, so that the char
> and block layers will use the global maps to lookup
> their devices.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> ---
>
> diff --git a/fs/Makefile b/fs/Makefile
> index 82b6ae1..a085706 100644
> --- a/fs/Makefile
> +++ b/fs/Makefile
> @@ -63,6 +63,8 @@ obj-y   += devpts/
>
> obj-$(CONFIG_PROFILING) += dcookies.o
> obj-$(CONFIG_DLM) += dlm/
> +
> +obj-$(CONFIG_CGROUP_DEVS) += devscontrol.o
>
> # Do not add any filesystems before this line
> obj-$(CONFIG_REISERFS_FS) += reiserfs/
> diff --git a/fs/devscontrol.c b/fs/devscontrol.c
> new file mode 100644
> index 0000000..ea282f3
> --- /dev/null
> +++ b/fs/devscontrol.c
> @@ -0,0 +1,291 @@
> +/*
> + * devscontrol.c - Device Controller
> + *
> + * Copyright 2007 OpenVZ SWsoft Inc
> + * Author: Pavel Emelyanov <xemul at openvz.org>
> + *
> + * This program is free software; you can redistribute it and/or modify
> + * it under the terms of the GNU General Public License as published by
> + * the Free Software Foundation; either version 2 of the License, or
> + * (at your option) any later version.

```
> + *
> + * This program is distributed in the hope that it will be useful,
> + * but WITHOUT ANY WARRANTY; without even the implied warranty of
> + * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
> + * GNU General Public License for more details.
> + */
> +
> +#include <linux/cgroup.h>
> +#include <linux/cdev.h>
> +#include <linux/err.h>
> +#include <linux/devscontrol.h>
> +#include <linux/uaccess.h>
> +#include <linux/fs.h>
> +#include <linux/genhd.h>
> +
> +struct devs_cgroup {
> + struct cgroup_subsys_state css;
> +
> + struct kobj_map *cdev_map;
> + struct kobj_map *bdev_map;
> +};
> +
> +static inline
> +struct devs_cgroup *css_to_devs(struct cgroup_subsys_state *css)
> +{
> + return container_of(css, struct devs_cgroup, css);
> +}
> +
> +static inline
> +struct devs_cgroup *cgroup_to_devs(struct cgroup *cont)
> +{
> + return css_to_devs(cgroup_subsys_state(cont, devs_subsys_id));
> +}
> +
> +struct kobj_map *task_cdev_map(struct task_struct *tsk)
> +{
> + struct cgroup_subsys_state *css;
> +
> + css = task_subsys_state(tsk, devs_subsys_id);
> + if (css->cgroup->parent == NULL)
> +  return NULL;
> + else
> +  return css_to_devs(css)->cdev_map;
> +}
> +
> +struct kobj_map *task_bdev_map(struct task_struct *tsk)
> +{
> + struct cgroup_subsys_state *css;
```

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

Thanks for working on this, Pavel.

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

> +
> +static void devs_destroy(struct cgroup_subsys *ss, struct cgroup *cont)
> +{
> + struct devs_cgroup *devs;
> +

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

```
> + else if (end[1] != '-')
> +  return -EINVAL;
> + if (end[2] == 'w')
> +  tmp |= FMODE_WRITE;
> + else if (end[2] != '-')
> +  return -EINVAL;
> +
> + *dev = MKDEV(major, minor);
> + *mode = tmp;
> + return 0;
> +}
> +
> +static int encode_perms_str(char *buf, int len, int chrdev, dev_t dev,
> +  int all, mode_t mode)
> +{
> + int ret;
> +
> + ret = snprintf(buf, len, "%c %d:", chrdev ? 'c' : 'b', MAJOR(dev));
> + if (all)
> +  ret += snprintf(buf + ret, len - ret, "*");
> + else
> +  ret += snprintf(buf + ret, len - ret, "%d", MINOR(dev));
> +
> + ret += snprintf(buf + ret, len - ret, " %c%c\n",
> +   (mode & FMODE_READ) ? 'r' : '-',
> +   (mode & FMODE_WRITE) ? 'w' : '-');
> +
> + return ret + 1;
> +}
> +
> +static ssize_t devs_write(struct cgroup *cont, struct cftype *cft,
> +  struct file *f, const char __user *ubuf,
> +  size_t nbytes, loff_t *pos)
> +{
> + int err, all, chrdev;
> + dev_t dev;
> + char buf[64];
> + struct devs_cgroup *devs;
> + mode_t mode;
```

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

```
> +
> + if (copy_from_user(buf, ubuf, sizeof(buf)))
> +  return -EFAULT;
> +
> + buf[sizeof(buf) - 1] = 0;
```

```
> + err = decode_perms_str(buf, &chrdev, &dev, &all, &mode);
> + if (err < 0)
> +  return err;
> +
> + devs = cgroup_to_devs(cont);
> +
> + if (mode == 0) {
> +  if (chrdev)
> +   err = cdev_del_from_map(devs->cdev_map, dev, all);
> +  else
> +   err = bdev_del_from_map(devs->bdev_map, dev, all);
> +
> +  if (err < 0)
> +   return err;
> +
> +  css_put(&devs->css);
> + } else {
> +  if (chrdev)
> +   err = cdev_add_to_map(devs->cdev_map, dev, all, mode);
> +  else
> +   err = bdev_add_to_map(devs->bdev_map, dev, all, mode);
> +
> +  if (err < 0)
> +   return err;
> +
> +  css_get(&devs->css);
> + }
> +
> + return nbytes;
> +}
> +
> +struct devs_dump_arg {
> + char *buf;
> + int pos;
> + int chrdev;
> +};
> +
> +static int devs_dump_one(dev_t dev, int range, mode_t mode, void *x)
> +{
> + struct devs_dump_arg *arg = x;
> + char tmp[64];
> + int len;
> +
> + len = encode_perms_str(tmp, sizeof(tmp), arg->chrdev, dev,
> +   range != 1, mode);
> +
> + if (arg->pos >= PAGE_SIZE - len)
> +  return 1;
```

```
> +
> + memcpy(arg->buf + arg->pos, tmp, len);
> + arg->pos += len;
> + return 0;
> +}
> +
> +static ssize_t devs_read(struct cgroup *cont, struct cftype *cft,
> +  struct file *f, char __user *ubuf, size_t nbytes, loff_t *pos)
> +{
> + struct devs_dump_arg arg;
> + struct devs_cgroup *devs;
> + ssize_t ret;
> +
> + arg.buf = (char *)__get_free_page(GFP_KERNEL);
> + if (arg.buf == NULL)
> +  return -ENOMEM;
> +
> + devs = cgroup_to_devs(cont);
> + arg.pos = 0;
> +
> + arg.chrdev = 1;
> + cdev_iterate_map(devs->cdev_map, devs_dump_one, &arg);
> +
> + arg.chrdev = 0;
> + bdev_iterate_map(devs->bdev_map, devs_dump_one, &arg);
> +
> + ret = simple_read_from_buffer(ubuf, nbytes, pos,
> +    arg.buf, arg.pos);
> +
> + free_page((unsigned long)arg.buf);
> + return ret;
> +}
> +
> +static struct cftype devs_files[] = {
> + {
> +  .name = "permissions",
> +  .write = devs_write,
> +  .read = devs_read,
> + },
> +};
> +
> +static int devs_populate(struct cgroup_subsys *ss, struct cgroup *cont)
> +{
> + return cgroup_add_files(cont, ss,
> +    devs_files, ARRAY_SIZE(devs_files));
> +}
> +
> +struct cgroup_subsys devs_subsys = {
```

> + .name = "devices",
> + .subsys_id = devs_subsys_id,
> + .create = devs_create,
> + .destroy = devs_destroy,
> + .populate = devs_populate,
> +};
> diff --git a/include/linux/cgroup_subsys.h b/include/linux/cgroup_subsys.h
> index 228235c..9c0cd2c 100644
> --- a/include/linux/cgroup_subsys.h
> +++ b/include/linux/cgroup_subsys.h
> @@ -42,3 +42,9 @@ SUBSYS(mem_cgroup)
> #endif
>
> /* */
> +
> +#ifdef CONFIG_CGROUP_DEVS
> +SUBSYS(devs)
> +#endif
> +
> +/* */
> diff --git a/include/linux/devscontrol.h b/include/linux/devscontrol.h
> new file mode 100644
> index 0000000..5f574e0
> --- /dev/null
> +++ b/include/linux/devscontrol.h
> @@ -0,0 +1,20 @@
> +#ifndef __DEVS_CONTROL_H__
> +#define __DEVS_CONTROL_H__
> +struct kobj_map;
> +struct task_struct;
> +
> +#ifdef CONFIG_CGROUP_DEVS
> +struct kobj_map *task_cdev_map(struct task_struct *);
> +struct kobj_map *task_bdev_map(struct task_struct *);
> +#else
> +static inline kobj_map *task_cdev_map(struct task_struct *tsk)
> +{
> + return NULL;
> +}
> +
> +static inline kobj_map *task_bdev_map(struct task_struct *tsk)
> +{
> + return NULL;
> +}
> +#endif
> +#endif
> diff --git a/init/Kconfig b/init/Kconfig
> index b886ac9..4dd53d6 100644

> --- a/init/Kconfig
> +++ b/init/Kconfig
> @@ -283,6 +283,12 @@ config CGROUP_DEBUG
>
>   Say N if unsure
>
> +config CGROUP_DEVS
> + bool "Devices cgroup subsystem"
> + depends on CGROUPS
> + help
> +   Controlls the visibility of devices
> +
> config CGROUP_NS
>   bool "Namespace cgroup subsystem"
>   depends on CGROUPS

_____

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

---

Subject: Re: [PATCH 0/4] Devices accessibility control group (v2)
Posted by Sukadev Bhattiprolu on Mon, 14 Jan 2008 17:42:20 GMT
View Forum Message <> Reply to Message

| > I started playing with this and noticed that even if I try to
| > enable read access to device [c, 1:3] it also grants access
| > to device [c, 1:5].
|
| Hm... I can't reproduce this:
|
| # /bin/echo 'c 1:3 r-' > /cnt/dev/0/devices.permissions
| # /bin/echo -n $$ > /cnt/dev/0/tasks
| # cat /cnt/dev/0/devices.permissions
| c 1:3 r-
| # hexdump /dev/null
| # hexdump /dev/zero
| hexdump: /dev/zero: No such device or address
| hexdump: /dev/zero: Bad file descriptor
|
| Maybe you have played with devs cgroups before getting this?
| Can you show what's the contents of the devices.permissions file
| in your case?

Here is the repro again. I even tried after a reboot. Basically,
granting access to /dev/null is also granting access to /dev/zero.

 # cat devices.permissions

```
# hexdump /dev/zero
hexdump: /dev/zero: No such device or address
hexdump: /dev/zero: Bad file descriptor
# hexdump /dev/null
hexdump: /dev/null: No such device or address
hexdump: /dev/null: Bad file descriptor
# echo 'c 1:3 r-' > devices.permissions
# hexdump /dev/null
# hexdump /dev/zero
0000000 0000 0000 0000 0000 0000 0000 0000 0000
*
^C
# cat tasks
3279
22266
# ps
  PID TTY          TIME CMD
 3279 pts/0    00:00:00 bash
22267 pts/0    00:00:00 ps
```

_____
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 (v2)
Posted by Paul Menage on Mon, 14 Jan 2008 21:18:49 GMT
View Forum Message <> Reply to Message

On Jan 8, 2008 1:02 AM, Pavel Emelyanov <xemul@openvz.org> wrote:
> * Add the support for devices ranges. I.e. someone might
> wish to tell smth like b 5:[0-10] r- to this subsystem.

I'd be inclined to leave support for that in userspace, rather than
adding fancier parsing to the kernel API.

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 Mon, 14 Jan 2008 21:54:33 GMT
View Forum Message <> Reply to Message

On Jan 8, 2008 1:18 AM, Pavel Emelyanov <xemul@openvz.org> wrote:
> Each new group will have its own maps for char and block
> layers. The devices access list is tuned via the
> devices.permissions file.
>
> One may read from the file to get the configured
> state. E.g.
>
> # cat /cont/devices/0/devices.permissions
> c 1:* rw
> b 8:1 rw
>
> The top container isn't initialized, so that the char
> and block layers will use the global maps to lookup
> their devices.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> ---
>
> diff --git a/fs/Makefile b/fs/Makefile
> index 82b6ae1..a085706 100644
> --- a/fs/Makefile
> +++ b/fs/Makefile
> @@ -63,6 +63,8 @@ obj-y   += devpts/
>
> obj-$(CONFIG_PROFILING)   += dcookies.o
> obj-$(CONFIG_DLM)   += dlm/
> +
> +obj-$(CONFIG_CGROUP_DEVS)   += devscontrol.o
>
> # Do not add any filesystems before this line
> obj-$(CONFIG_REISERFS_FS)   += reiserfs/
> diff --git a/fs/devscontrol.c b/fs/devscontrol.c
> new file mode 100644
> index 0000000..ea282f3
> --- /dev/null
> +++ b/fs/devscontrol.c
> @@ -0,0 +1,291 @@
> +/*
> + * devscontrol.c - Device Controller
> + *
> + * Copyright 2007 OpenVZ SWsoft Inc
> + * Author: Pavel Emelyanov <xemul at openvz.org>
> + *
> + * This program is free software; you can redistribute it and/or modify
> + * it under the terms of the GNU General Public License as published by
> + * the Free Software Foundation; either version 2 of the License, or

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

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

make this

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

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

```
> +       arg.buf = (char *)__get_free_page(GFP_KERNEL);
> +       if (arg.buf == NULL)
> +               return -ENOMEM;
> +
> +       devs = cgroup_to_devs(cont);
> +       arg.pos = 0;
> +
> +       arg.chrdev = 1;
> +       cdev_iterate_map(devs->cdev_map, devs_dump_one, &arg);
```

> +
> +	arg.chrdev = 0;
> +	bdev_iterate_map(devs->bdev_map, devs_dump_one, &arg);

Is there any chance of this overflowing the buffer page?

Paul

_____
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, 15 Jan 2008 07:53:13 GMT
View Forum Message <> Reply to Message

[snip]

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

Right. When you create a new cgroup you have an empty perms
set. Maybe it's worth inheriting the perms from the parent
container, but I think that empty set is better as you will
reconfigure it anyway.

[snip]

>> +static ssize_t devs_write(struct cgroup *cont, struct cftype *cft,
>> + struct file *f, const char __user *ubuf,
>> + size_t nbytes, loff_t *pos)
>> +{
>> + int err, all, chrdev;
>> + dev_t dev;
>> + char buf[64];
>> + struct devs_cgroup *devs;
>> + mode_t mode;
>
> (Of course this will require some privilege, i assume that's a detail
> you'll add next time around)

Hm... I though that privileges are governed at the cgroup level.... No?

[snip]

_____

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, 15 Jan 2008 07:58:48 GMT
View Forum Message <> Reply to Message

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

```
>> index 0000000..ea282f3
>> --- /dev/null
>> +++ b/fs/devscontrol.c
>> @@ -0,0 +1,291 @@
>> +/*
>> + * devscontrol.c - Device Controller
>> + *
>> + * Copyright 2007 OpenVZ SWsoft Inc
>> + * Author: Pavel Emelyanov <xemul at openvz.org>
>> + *
>> + * This program is free software; you can redistribute it and/or modify
>> + * it under the terms of the GNU General Public License as published by
>> + * the Free Software Foundation; either version 2 of the License, or
>> + * (at your option) any later version.
>> + *
>> + * This program is distributed in the hope that it will be useful,
>> + * but WITHOUT ANY WARRANTY; without even the implied warranty of
>> + * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
>> + * GNU General Public License for more details.
>> + */
>> +
>> +#include <linux/cgroup.h>
>> +#include <linux/cdev.h>
>> +#include <linux/err.h>
>> +#include <linux/devscontrol.h>
>> +#include <linux/uaccess.h>
>> +#include <linux/fs.h>
>> +#include <linux/genhd.h>
>> +
>> +struct devs_cgroup {
>> +       struct cgroup_subsys_state css;
>> +
>> +       struct kobj_map *cdev_map;
>> +       struct kobj_map *bdev_map;
>> +};
>> +
>> +static inline
>> +struct devs_cgroup *css_to_devs(struct cgroup_subsys_state *css)
>> +{
>> +       return container_of(css, struct devs_cgroup, css);
>> +}
>> +
>> +static inline
>> +struct devs_cgroup *cgroup_to_devs(struct cgroup *cont)
>> +{
>> +       return css_to_devs(cgroup_subsys_state(cont, devs_subsys_id));
>> +}
>> +
```

```
>> +struct kobj_map *task_cdev_map(struct task_struct *tsk)
>> +{
>> +	struct cgroup_subsys_state *css;
>> +
>> +	css = task_subsys_state(tsk, devs_subsys_id);
>> +	if (css->cgroup->parent == NULL)
>> +		return NULL;
>> +	else
>> +		return css_to_devs(css)->cdev_map;
>> +}
>
> For this (and task_bdev_map) it would be cleaner to just leave the
> cdev_map and bdev_map in the root cgroup as NULL, so you could just
> make this
>
> struct kobj_map *task_cdev_map(struct task_struct *tsk)
> {
>   return css_to_devs(task_subsys_state(tsk, devs_subsys_id))->cdev_map;
> }
```

Ok, thanks, I'll make it in the next version.

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

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

Yup. The locking inside this call is unnecessary as maps
are atomic by themselves. I'll add the comment about it.

```
>> +	arg.buf = (char *)__get_free_page(GFP_KERNEL);
>> +	if (arg.buf == NULL)
>> +		return -ENOMEM;
>> +
>> +	devs = cgroup_to_devs(cont);
>> +	arg.pos = 0;
>> +
>> +	arg.chrdev = 1;
>> +	cdev_iterate_map(devs->cdev_map, devs_dump_one, &arg);
>> +
>> +	arg.chrdev = 0;
>> +	bdev_iterate_map(devs->bdev_map, devs_dump_one, &arg);
>
```

> Is there any chance of this overflowing the buffer page?

Well, I have checks that we've filled all the page and do not
go on dumping the info in this case. So the only bad thing user
can have is that he doesn't see the full perms list, but the
kernel won't OOPs ;) Since we have from 8 to 16 bytes per perm
line, we may have from 256 to 256 permissions set for cgroup.

Not that much, but that's enough for a first time (I haven't seen
any OpenVZ user who sets more than 50 devperms). But sure this
is one of TODO-s for the next patch versions.

> Paul
>

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

---

Subject: Re: [PATCH 2/4] The character devices layer changes
Posted by Pavel Emelianov on Tue, 15 Jan 2008 08:05:22 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelyanov (xemul@openvz.org):
>> 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.
>>
>> To address this issue, I store the last_perm and last(_map)
>> fields on the struct cdev (and protect them with the cdev_lock)
>> and force the re-lookup in the kobj mappings if needed.
>>
>> I know, this might be slow, but I have two points for it:
>> 1. The re-lookup happens on open() only which is not
>>   a fast-path. Besides, this is so for block layer and
>>   nobody complains;
>> 2. On a well-isolated setup, when each container has its
>>   own filesystem this is no longer a problem - each
>>   cgroup will cache the cdev on its inode and work good.
>
> What about simply returning -EPERM when open()ing a cdev
> with ->map!=task_cdev_map(current)?

In this case it will HAVE to setup isolated filesystem for
each cgroup. I thought that this flexibility doesn't hurt.

> Shouldn't be a problem for ttys, since the container init
> already has the tty open, right?

Yup, but this is not the case for /dev/null or /dev/zero.

> Otherwise, the patchset looks good to me. Want to look
> through this one a little more (i think that'd be easier
> with the -EPERM approach) and scrutinize patch 4, but
> overall it makes sense.

OK, thanks.

> If I understand right, we're taking 14k per cgroup for
> kobjmaps? Do we consider that a problem?

14k? I allocate the struct kobj_map which is only 256 pointers
(i.e. - 2K) and the struct probe that is 32 bytes. I.e. 4k
or a single page. I think this is OK.

> thanks,
> -serge
>

[snip]

_____
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 (v2)
Posted by Pavel Emelianov on Tue, 15 Jan 2008 08:06:02 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On Jan 8, 2008 1:02 AM, Pavel Emelyanov <xemul@openvz.org> wrote:
>> * Add the support for devices ranges. I.e. someone might
>> wish to tell smth like b 5:[0-10] r- to this subsystem.
>
> I'd be inclined to leave support for that in userspace, rather than
> adding fancier parsing to the kernel API.

Yup. I agree with that.

> Paul
>

Thanks,
Pavel

_____
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 (v2)
Posted by Pavel Emelianov on Tue, 15 Jan 2008 08:22:03 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> | > I started playing with this and noticed that even if I try to
> | > enable read access to device [c, 1:3] it also grants access
> | > to device [c, 1:5].
> |
> | Hm... I can't reproduce this:
> |

> | # /bin/echo 'c 1:3 r-' > /cnt/dev/0/devices.permissions
> | # /bin/echo -n $$ > /cnt/dev/0/tasks
> | # cat /cnt/dev/0/devices.permissions
> | c 1:3 r-
> | # hexdump /dev/null
> | # hexdump /dev/zero
> | hexdump: /dev/zero: No such device or address
> | hexdump: /dev/zero: Bad file descriptor
> |
> | Maybe you have played with devs cgroups before getting this?
> | Can you show what's the contents of the devices.permissions file
> | in your case?
>
> Here is the repro again. I even tried after a reboot. Basically,
> granting access to /dev/null is also granting access to /dev/zero.
>
> # cat devices.permissions
> # hexdump /dev/zero
> hexdump: /dev/zero: No such device or address
> hexdump: /dev/zero: Bad file descriptor
> # hexdump /dev/null
> hexdump: /dev/null: No such device or address
> hexdump: /dev/null: Bad file descriptor
> # echo 'c 1:3 r-' > devices.permissions
> # hexdump /dev/null
> # hexdump /dev/zero
> 0000000 0000 0000 0000 0000 0000 0000 0000 0000
> *
> ^C
> # cat tasks
> 3279
> 22266
> # ps
>   PID TTY   TIME CMD
>   3279 pts/0   00:00:00 bash
> 22267 pts/0   00:00:00 ps
>

This all looks completely incomprehensible :(

Here's my test:
mount -t cgroup none /cnt/dev/ -o devices
mkdir /cnt/dev/0
/bin/echo -n $$ > /cnt/dev/0/tasks
cat /cnt/dev/0/devices.permissions
hexdump /dev/zero
hexdump: /dev/zero: No such device or address
hexdump: /dev/zero: Bad file descriptor

hexdump /dev/null
hexdump: /dev/null: No such device or address
hexdump: /dev/null: Bad file descriptor
echo 'c 1:3 r-' > /cnt/dev/0/devices.permissions
cat /cnt/dev/0/devices.permissions
c 1:3 r-
hexdump /dev/null
hexdump /dev/zero
hexdump: /dev/zero: No such device or address
hexdump: /dev/zero: Bad file descriptor

Sukadev, could you please try to track the problem as you
seem to be the only person who's experiencing problems
with that.

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 Tue, 15 Jan 2008 14:44:40 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> [snip]
>
> > Thanks for working on this, Pavel.
> >
> > My only question with this patch is - so if I create a devs
> > cgroup which only has access to, say /dev/loop0 and /dev/tty3,
> > and someone in that cgroup manages to create a new cgroup, the
> > new cgroup will have all the default permissions again, rather
> > than inherit the permissions from this cgroup, right?
>
> Right. When you create a new cgroup you have an empty perms
> set. Maybe it's worth inheriting the perms from the parent
> container, but I think that empty set is better as you will
> reconfigure it anyway.
>
> [snip]
>
> >> +static ssize_t devs_write(struct cgroup *cont, struct cftype *cft,
> >> + struct file *f, const char __user *ubuf,

> >> + size_t nbytes, loff_t *pos)
> >> +{
> >> + int err, all, chrdev;
> >> + dev_t dev;
> >> + char buf[64];
> >> + struct devs_cgroup *devs;
> >> + mode_t mode;
> > >
> > > (Of course this will require some privilege, i assume that's a detail
> > > you'll add next time around)
> >
> > Hm... I though that privileges are governed at the cgroup level.... No?

I don't think so... Wouldn't really make sense for the cgroup
infrastructure to presume to know what to enforce, and I don't see any
checks around the _write functions in cgroup.c, and no capable() calls
at all.

-serge

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

---

Subject: Re: [PATCH 2/4] The character devices layer changes
Posted by serue on Tue, 15 Jan 2008 14:54:48 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> Serge E. Hallyn wrote:
> > Quoting Pavel Emelyanov (xemul@openvz.org):
> >> 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.
> >>
> >> To address this issue, I store the last_perm and last(_map)
> >> fields on the struct cdev (and protect them with the cdev_lock)
> >> and force the re-lookup in the kobj mappings if needed.
> >>
> >> I know, this might be slow, but I have two points for it:
> >> 1. The re-lookup happens on open() only which is not
> >>   a fast-path. Besides, this is so for block layer and

> >>   nobody complains;
> >> 2. On a well-isolated setup, when each container has its
> >>   own filesystem this is no longer a problem - each
> >>   cgroup will cache the cdev on its inode and work good.
> >
> > What about simply returning -EPERM when open()ing a cdev
> > with ->map!=task_cdev_map(current)?
>
> In this case it will HAVE to setup isolated filesystem for
> each cgroup. I thought that this flexibility doesn't hurt.

The cost and effort of setting up a private /dev seems so minimal to me
it seems worth not dealing with the inode->map switching around.

But maybe that's just me.

> > Shouldn't be a problem for ttys, since the container init
> > already has the tty open, right?
>
> Yup, but this is not the case for /dev/null or /dev/zero.
>
> > Otherwise, the patchset looks good to me. Want to look
> > through this one a little more (i think that'd be easier
> > with the -EPERM approach) and scrutinize patch 4, but
> > overall it makes sense.
>
> OK, thanks.
>
> > If I understand right, we're taking 14k per cgroup for
> > kobjmaps? Do we consider that a problem?
>
> 14k? I allocate the struct kobj_map which is only 256 pointers
> (i.e. - 2K) and the struct probe that is 32 bytes. I.e. 4k
> or a single page. I think this is OK.

Oops, I was thinking the probes were all pre-allocated. Sorry.

> > thanks,
> > -serge
> >
>
> [snip]

_____
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 Tue, 15 Jan 2008 16:13:40 GMT
View Forum Message <> Reply to Message

On Jan 15, 2008 6:44 AM, Serge E. Hallyn <serue@us.ibm.com> wrote:
>
> I don't think so... Wouldn't really make sense for the cgroup
> infrastructure to presume to know what to enforce, and I don't see any
> checks around the _write functions in cgroup.c, and no capable() calls
> at all.

The cgroup filesystem can provide simple unix-level permissions on any given file. Am I right in thinking that having an entry in the mapper doesn't automatically give privileges for a device to the members of the cgroup, but they also have to have sufficient privilege in their own right? If so, that might be sufficient.

One other thought - should the parse/print routines themselves do a translation based on the device mappings for the writer/reader's cgroup? That way you could safely give a VE full permission to write to its children's device maps, but it would only be able to add/remap device targets that it could address itself.

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 serue on Tue, 15 Jan 2008 17:49:41 GMT
View Forum Message <> Reply to Message

Quoting Paul Menage (menage@google.com):
> On Jan 15, 2008 6:44 AM, Serge E. Hallyn <serue@us.ibm.com> wrote:
> >
> > I don't think so... Wouldn't really make sense for the cgroup
> > infrastructure to presume to know what to enforce, and I don't see any
> > checks around the _write functions in cgroup.c, and no capable() calls
> > at all.
>
> The cgroup filesystem can provide simple unix-level permissions on any
> given file. Am I right in thinking that having an entry in the mapper
> doesn't automatically give privileges for a device to the members of
> the cgroup, but they also have to have sufficient privilege in their
> own right? If so, that might be sufficient.

Oh, well actually I think what we'd want is to require both
CAP_NS_OVERRIDE and either CAP_MKNOD or CAP_SYS_ADMIN. So it's probably
fine to leave this as is for now, and after I resend the patchset which
pushes CAP_NS_OVERRIDE (which is in a 4-patch userns patchset I've
been sitting on) the extra checks can be added.

> One other thought - should the parse/print routines themselves do a
> translation based on the device mappings for the writer/reader's
> cgroup? That way you could safely give a VE full permission to write
> to its children's device maps, but it would only be able to add/remap
> device targets that it could address itself.

Oh, well if we do this then we can just as well use the translation
functions to not allow a VE to add to its own set of devices, right?

Then maybe capable(CAP_NS_OVERRIDE|CAP_SYS_ADMIN) would only be required
to add devices.

Though there *is* some bit of danger to removing devices from a
privileged daemon, isn't there? Though I can't think of examples
just now. (Sorry, piercing headache, can't think quite right, will
think about this later)

-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 Tue, 15 Jan 2008 17:54:27 GMT
View Forum Message <> Reply to Message

On Jan 15, 2008 9:49 AM, Serge E. Hallyn <serue@us.ibm.com> wrote:
> > One other thought - should the parse/print routines themselves do a
> > translation based on the device mappings for the writer/reader's
> > cgroup? That way you could safely give a VE full permission to write
> > to its children's device maps, but it would only be able to add/remap
> > device targets that it could address itself.
>
> Oh, well if we do this then we can just as well use the translation
> functions to not allow a VE to add to its own set of devices, right?

Right.

>
> Then maybe capable(CAP_NS_OVERRIDE|CAP_SYS_ADMIN) would only be required

> to add devices.

Or simply require that they be added by someone who already has access
to that device via their own control group? The root cgroup would have
access to all devices.

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 serue on Tue, 15 Jan 2008 18:17:17 GMT
View Forum Message <> Reply to Message

Quoting Paul Menage (menage@google.com):
> On Jan 15, 2008 9:49 AM, Serge E. Hallyn <serue@us.ibm.com> wrote:
> > > One other thought - should the parse/print routines themselves do a
> > > translation based on the device mappings for the writer/reader's
> > > cgroup? That way you could safely give a VE full permission to write
> > > to its children's device maps, but it would only be able to add/remap
> > > device targets that it could address itself.
> >
> > Oh, well if we do this then we can just as well use the translation
> > functions to not allow a VE to add to its own set of devices, right?
>
> Right.
>
> >
> > Then maybe capable(CAP_NS_OVERRIDE|CAP_SYS_ADMIN) would only be required
> > to add devices.
>
> Or simply require that they be added by someone who already has access
> to that device via their own control group? The root cgroup would have
> access to all devices.

Where by 'have access' you mean access to create the device? That
sounds good.

thanks,
-serge

_____
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 (v2)
Posted by Sukadev Bhattiprolu on Thu, 17 Jan 2008 06:26:05 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@openvz.org] wrote:
| sukadev@us.ibm.com wrote:
| > | > I started playing with this and noticed that even if I try to
| > | > enable read access to device [c, 1:3] it also grants access
| > | > to device [c, 1:5].
| > |
| > | Hm... I can't reproduce this:
| > |
| > | # /bin/echo 'c 1:3 r-' > /cnt/dev/0/devices.permissions
| > | # /bin/echo -n $$ > /cnt/dev/0/tasks
| > | # cat /cnt/dev/0/devices.permissions
| > | c 1:3 r-
| > | # hexdump /dev/null
| > | # hexdump /dev/zero
| > | hexdump: /dev/zero: No such device or address
| > | hexdump: /dev/zero: Bad file descriptor
| > |
| > | Maybe you have played with devs cgroups before getting this?
| > | Can you show what's the contents of the devices.permissions file
| > | in your case?
| >
| > Here is the repro again. I even tried after a reboot. Basically,
| > granting access to /dev/null is also granting access to /dev/zero.
| >
| > # cat devices.permissions
| > # hexdump /dev/zero
| > hexdump: /dev/zero: No such device or address
| > hexdump: /dev/zero: Bad file descriptor
| > # hexdump /dev/null
| > hexdump: /dev/null: No such device or address
| > hexdump: /dev/null: Bad file descriptor
| > # echo 'c 1:3 r-' > devices.permissions
| > # hexdump /dev/null
| > # hexdump /dev/zero
| > 0000000 0000 0000 0000 0000 0000 0000 0000 0000
| > *
| > ^C
| > # cat tasks
| > 3279
| > 22266
| > # ps
| >   PID TTY   TIME CMD
| >   3279 pts/0   00:00:00 bash
| > 22267 pts/0   00:00:00 ps
| >

|
| This all looks completely incomprehensible :(
|
| Here's my test:
| # mount -t cgroup none /cnt/dev/ -o devices
| # mkdir /cnt/dev/0
| # /bin/echo -n $$ > /cnt/dev/0/tasks
| # cat /cnt/dev/0/devices.permissions
| # hexdump /dev/zero
| hexdump: /dev/zero: No such device or address
| hexdump: /dev/zero: Bad file descriptor

Can you try this sequence:

 - grant access to /dev/zero,
 - hexdump /dev/zero
 - revoke access to /dev/zero
 - hexdump /dev/null
 - hexdump /dev/zero.

| # hexdump /dev/null
| hexdump: /dev/null: No such device or address
| hexdump: /dev/null: Bad file descriptor
| # echo 'c 1:3 r-' > /cnt/dev/0/devices.permissions
| # cat /cnt/dev/0/devices.permissions
| c 1:3 r-
| # hexdump /dev/null
| # hexdump /dev/zero
| hexdump: /dev/zero: No such device or address
| hexdump: /dev/zero: Bad file descriptor
|
|
| Sukadev, could you please try to track the problem as you
| seem to be the only person who's experiencing problems
| with that.

I suspect the 'caching' of the last_mode (that you introduce in PATCH 2/4)
combined with the fact that /dev/zero, /dev/null, /dev/kmem etc share
a _SINGLE_ 'struct cdev' leads to the problem I am running into with
/dev/zero and /dev/null.

Here is a what I suspect is happening (sorry, for low-level details)

Following sequence seems to repro it consistently for me:

 $ mount -t cgroup none /container/devs/ -o devices
 $ mkdir /container/devs/0

```
$ cd !$
cd /container/devs/0
$ echo $$ > tasks

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

$ hexdump /dev/null
hexdump: /dev/null: No such device or address
hexdump: /dev/null: Bad file descriptor

$ echo 'c 1:3 r-' > devices.permissions

$ hexdump /dev/null

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

No surprise so far.

```
$ echo 'c 1:5 r-' > devices.permissions
$ hexdump /dev/zero
0000000 0000 0000 0000 0000 0000 0000 0000 0000
*
^C
```

Now grant read access to /dev/zero and more importantly, create a properly initialized inode for it.

```
$ echo 'c 1:5 --' > devices.permissions
```

Then remove access to /dev/zero. This removes the kobject for /dev/zero from map. Also cdev_map_reset() sets cdev->last to NULL.

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

Since cdev->last is NULL, chrdev_open() calls kobj_lookup() which returns a NULL kobj and the open fails.

```
$ hexdump /dev/null # XXX
```

Again, since cdev->last is NULL, kobj_lookup() is called, this time for /dev/null. This succeeds and cdev->last is correctly initialized.
Eventually this open of /dev/null succeeds.

$ hexdump /dev/zero
0000000 0000 0000 0000 0000 0000 0000 0000 0000

Now the open of /dev/zero also succeeds !

I suspect that the reason is that when we first successfully read /dev/zero, we created/initialized an inode for it. This inode has the inode->i_cdev set correctly.

By reading /dev/null (marked XXX above), cdev->last is also correctly set.

But since /dev/zero and /dev/null _SHARE_ a 'struct cdev', when we call chrdev_open() for /dev/zero, we check the permissions of this common cdev and grant /dev/zero the same permissions as /dev/null.

I suspect we will get this behavior with all devices implemented by the 'mem' driver in drivers/char/mem.c. I was able to repro with /dev/full [c, 1:7])

Sukadev

_____
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 (v2)
Posted by Pavel Emelianov on Mon, 21 Jan 2008 08:31:45 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> Pavel Emelianov [xemul@openvz.org] wrote:
> | sukadev@us.ibm.com wrote:
> | > | > I started playing with this and noticed that even if I try to
> | > | > enable read access to device [c, 1:3] it also grants access
> | > | > to device [c, 1:5].
> | > |
> | > | Hm... I can't reproduce this:
> | > |
> | > | # /bin/echo 'c 1:3 r-' > /cnt/dev/0/devices.permissions
> | > | # /bin/echo -n $$ > /cnt/dev/0/tasks
> | > | # cat /cnt/dev/0/devices.permissions
> | > | c 1:3 r-
> | > | # hexdump /dev/null
> | > | # hexdump /dev/zero
> | > | hexdump: /dev/zero: No such device or address
> | > | hexdump: /dev/zero: Bad file descriptor

> | > |
> | > | Maybe you have played with devs cgroups before getting this?
> | > | Can you show what's the contents of the devices.permissions file
> | > | in your case?
> | >
> | > Here is the repro again. I even tried after a reboot. Basically,
> | > granting access to /dev/null is also granting access to /dev/zero.
> | >
> | > # cat devices.permissions
> | > # hexdump /dev/zero
> | > hexdump: /dev/zero: No such device or address
> | > hexdump: /dev/zero: Bad file descriptor
> | > # hexdump /dev/null
> | > hexdump: /dev/null: No such device or address
> | > hexdump: /dev/null: Bad file descriptor
> | > # echo 'c 1:3 r-' > devices.permissions
> | > # hexdump /dev/null
> | > # hexdump /dev/zero
> | > 0000000 0000 0000 0000 0000 0000 0000 0000 0000
> | > *
> | > ^C
> | > # cat tasks
> | > 3279
> | > 22266
> | > # ps
> | >   PID TTY   TIME CMD
> | >   3279 pts/0   00:00:00 bash
> | > 22267 pts/0   00:00:00 ps
> | >
> |
> | This all looks completely incomprehensible :(
> |
> | Here's my test:
> | # mount -t cgroup none /cnt/dev/ -o devices
> | # mkdir /cnt/dev/0
> | # /bin/echo -n $$ > /cnt/dev/0/tasks
> | # cat /cnt/dev/0/devices.permissions
> | # hexdump /dev/zero
> | hexdump: /dev/zero: No such device or address
> | hexdump: /dev/zero: Bad file descriptor
>
> Can you try this sequence:
>
> - grant access to /dev/zero,
> - hexdump /dev/zero
> - revoke access to /dev/zero
> - hexdump /dev/null
> - hexdump /dev/zero.

OK, I'll try it, thanks.

> | # hexdump /dev/null
> | hexdump: /dev/null: No such device or address
> | hexdump: /dev/null: Bad file descriptor
> | # echo 'c 1:3 r-' > /cnt/dev/0/devices.permissions
> | # cat /cnt/dev/0/devices.permissions
> | c 1:3 r-
> | # hexdump /dev/null
> | # hexdump /dev/zero
> | hexdump: /dev/zero: No such device or address
> | hexdump: /dev/zero: Bad file descriptor
> |
> |
> | Sukadev, could you please try to track the problem as you
> | seem to be the only person who's experiencing problems
> | with that.
>
>
> I suspect the 'caching' of the last_mode (that you introduce in PATCH 2/4)
> combined with the fact that /dev/zero, /dev/null, /dev/kmem etc share
> a _SINGLE_ 'struct cdev' leads to the problem I am running into with
> /dev/zero and /dev/null.
>
> Here is a what I suspect is happening (sorry, for low-level details)
>
> Following sequence seems to repro it consistently for me:
>
> $ mount -t cgroup none /container/devs/ -o devices
> $ mkdir /container/devs/0
> $ cd !$
> cd /container/devs/0
> $ echo $$ > tasks
>
> $ hexdump /dev/zero
> hexdump: /dev/zero: No such device or address
> hexdump: /dev/zero: Bad file descriptor
>
> $ hexdump /dev/null
> hexdump: /dev/null: No such device or address
> hexdump: /dev/null: Bad file descriptor
>
> $ echo 'c 1:3 r-' > devices.permissions
>
> $ hexdump /dev/null
>
> $ hexdump /dev/zero

> hexdump: /dev/zero: No such device or address
> hexdump: /dev/zero: Bad file descriptor
>
> No surprise so far.
>
> $ echo 'c 1:5 r-' > devices.permissions
> $ hexdump /dev/zero
> 0000000 0000 0000 0000 0000 0000 0000 0000 0000
> *
> ^C
>
> Now grant read access to /dev/zero and more importantly, create a properly
> initialized inode for it.
>
> $ echo 'c 1:5 --' > devices.permissions
>
> Then remove access to /dev/zero. This removes the kobject for /dev/zero from
> map. Also cdev_map_reset() sets cdev->last to NULL.
>
> $ hdz
> hexdump: /dev/zero: No such device or address
> hexdump: /dev/zero: Bad file descriptor
>
> Since cdev->last is NULL, chrdev_open() calls kobj_lookup() which returns a
> NULL kobj and the open fails.
>
> $ hexdump /dev/null # XXX
>
> Again, since cdev->last is NULL, kobj_lookup() is called, this time for
> /dev/null. This succeeds and cdev->last is correctly initialized.
> Eventually this open of /dev/null succeeds.
>
> $ hexdump /dev/zero
> 0000000 0000 0000 0000 0000 0000 0000 0000 0000
>
> Now the open of /dev/zero also succeeds !

Hm... The analysis looks correct. Thanks, Sukadev, I'll try
to resolve this issue.

> I suspect that the reason is that when we first successfully read /dev/zero,
> we created/initialized an inode for it. This inode has the inode->i_cdev set
> correctly.
>
> By reading /dev/null (marked XXX above), cdev->last is also correctly set.
>
> But since /dev/zero and /dev/null _SHARE_ a 'struct cdev', when we call
> chrdev_open() for /dev/zero, we check the permissions of this common cdev

> and grant /dev/zero the same permissions as /dev/null.
>
> I suspect we will get this behavior with all devices implemented by
> the 'mem' driver in drivers/char/mem.c. I was able to repro with
> /dev/full [c, 1:7])
>
> Sukadev
>

_____

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