Subject: [PATCH 0/8] CGroup Files: Add write_string control file method
Posted by Paul Menage on Fri, 20 Jun 2008 23:43:58 GMT
View Forum Message <> Reply to Message

This is a resend of a patchset that I sent last month, reworked to
remove some controversial locking proposals. All locking is now explicit.

This patchset provides:

1) A new write_string() cgroup file method that copies the user's data
to kernel space and invokes the relevant handler with the
nul-terminated kernelspace buffer

2) A new helper function, cgroup_lock_live_group(), which combines
taking the cgroup lock and checking the liveness of a cgroup, to allow
simplification of a common lock/check idiom in cgroup file handlers.

3) Conversion of several raw write handlers in cgroup, cpuset,
devcgroup and res_counter to use typed handlers and the new locking
specifications.

Signed-off-by: Paul Menage <menage@google.com>

--

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

Subject: [PATCH 2/8] CGroup Files: Add write_string cgroup control file method
Posted by Paul Menage on Fri, 20 Jun 2008 23:44:00 GMT
View Forum Message <> Reply to Message

This patch adds a write_string() method for cgroups control files. The
semantics are that a buffer is copied from userspace to kernelspace
and the handler function invoked on that buffer. The buffer is
guaranteed to be nul-terminated, and no longer than max_write_len
(defaulting to 64 bytes if unspecified). Later patches will convert
existing raw file write handlers in control group subsystems to use
this method.

Signed-off-by: Paul Menage <menage@google.com>

---
 include/linux/cgroup.h | 14 +++++++++++++
 kernel/cgroup.c | 35 +++++++++++++++++++++++++++++++++++
 2 files changed, 49 insertions(+)

```
Index: cws-2.6.26-rc5-mm3/include/linux/cgroup.h
===================================================================
--- cws-2.6.26-rc5-mm3.orig/include/linux/cgroup.h
+++ cws-2.6.26-rc5-mm3/include/linux/cgroup.h
@@ -205,6 +205,13 @@ struct cftype {
   * subsystem, followed by a period */
  char name[MAX_CFTYPE_NAME];
  int private;
+
+ /*
+  * If non-zero, defines the maximum length of string that can
+  * be passed to write_string; defaults to 64
+  */
+ size_t max_write_len;
+
  int (*open)(struct inode *inode, struct file *file);
  ssize_t (*read)(struct cgroup *cgrp, struct cftype *cft,
    struct file *file,
@@ -249,6 +256,13 @@ struct cftype {
  int (*write_s64)(struct cgroup *cgrp, struct cftype *cft, s64 val);

  /*
+  * write_string() is passed a nul-terminated kernelspace
+  * buffer of maximum length determined by max_write_len.
+  * Returns 0 or -ve error code.
+  */
+ int (*write_string)(struct cgroup *cgrp, struct cftype *cft,
+       const char *buffer);
+ /*
   * trigger() callback can be used to get some kick from the
   * userspace, when the actual string written is not important
   * at all. The private field can be used to determine the
Index: cws-2.6.26-rc5-mm3/kernel/cgroup.c
===================================================================
--- cws-2.6.26-rc5-mm3.orig/kernel/cgroup.c
+++ cws-2.6.26-rc5-mm3/kernel/cgroup.c
@@ -1363,6 +1363,39 @@ static ssize_t cgroup_write_X64(struct c
  return retval;
 }

+static ssize_t cgroup_write_string(struct cgroup *cgrp, struct cftype *cft,
+       struct file *file,
+       const char __user *userbuf,
+       size_t nbytes, loff_t *unused_ppos)
+{
+ char local_buffer[64];
+ int retval = 0;
```

```
+ size_t max_bytes = cft->max_write_len;
+ char *buffer = local_buffer;
+
+ if (!max_bytes)
+  max_bytes = sizeof(local_buffer) - 1;
+ if (nbytes >= max_bytes)
+  return -E2BIG;
+ /* Allocate a dynamic buffer if we need one */
+ if (nbytes >= sizeof(local_buffer)) {
+  buffer = kmalloc(nbytes + 1, GFP_KERNEL);
+  if (buffer == NULL)
+   return -ENOMEM;
+ }
+ if (nbytes && copy_from_user(buffer, userbuf, nbytes))
+  return -EFAULT;
+
+ buffer[nbytes] = 0;     /* nul-terminate */
+ strstrip(buffer);
+ retval = cft->write_string(cgrp, cft, buffer);
+ if (!retval)
+  retval = nbytes;
+ if (buffer != local_buffer)
+  kfree(buffer);
+ return retval;
+}
+
 static ssize_t cgroup_common_file_write(struct cgroup *cgrp,
         struct cftype *cft,
         struct file *file,
@@ -1440,6 +1473,8 @@ static ssize_t cgroup_file_write(struct
   return cft->write(cgrp, cft, file, buf, nbytes, ppos);
  if (cft->write_u64 || cft->write_s64)
   return cgroup_write_X64(cgrp, cft, file, buf, nbytes, ppos);
+ if (cft->write_string)
+  return cgroup_write_string(cgrp, cft, file, buf, nbytes, ppos);
  if (cft->trigger) {
   int ret = cft->trigger(cgrp, (unsigned int)cft->private);
   return ret ? ret : nbytes;

--
```

Subject: [PATCH 3/8] CGroup Files: Move the release_agent file to use typed

handlers
Posted by Paul Menage on Fri, 20 Jun 2008 23:44:01 GMT
View Forum Message <> Reply to Message

Adds cgroup_release_agent_write() and cgroup_release_agent_show()
methods to handle writing/reading the path to a cgroup hierarchy's
release agent. As a result, cgroup_common_file_read() is now unnecessary.

As part of the change, a previously-tolerated race in
cgroup_release_agent() is avoided by copying the current
release_agent_path prior to calling call_usermode_helper().

Signed-off-by: Paul Menage <menage@google.com>

---
 include/linux/cgroup.h | 2
 kernel/cgroup.c | 125 ++++++++++++++++++++++++++++++++----------------
 2 files changed, 59 insertions(+), 68 deletions(-)

Index: cws-2.6.26-rc5-mm3/kernel/cgroup.c
===
--- cws-2.6.26-rc5-mm3.orig/kernel/cgroup.c
+++ cws-2.6.26-rc5-mm3/kernel/cgroup.c
@@ -89,11 +89,7 @@ struct cgroupfs_root {
  /* Hierarchy-specific flags */
  unsigned long flags;

- /* The path to use for release notifications. No locking
- * between setting and use - so if userspace updates this
- * while child cgroups exist, you could miss a
- * notification. We ensure that it's always a valid
- * NUL-terminated string */
+ /* The path to use for release notifications. */
  char release_agent_path[PATH_MAX];
 };

@@ -1329,6 +1325,45 @@ enum cgroup_filetype {
  FILE_RELEASE_AGENT,
 };

+/**
+ * cgroup_lock_live_group - take cgroup_mutex and check that cgrp is alive.
+ * @cgrp: the cgroup to be checked for liveness
+ *
+ * Returns true (with lock held) on success, or false (with no lock
+ * held) on failure.
+ */
+int cgroup_lock_live_group(struct cgroup *cgrp)
+{

```
+ mutex_lock(&cgroup_mutex);
+ if (cgroup_is_removed(cgrp)) {
+  mutex_unlock(&cgroup_mutex);
+  return false;
+ }
+ return true;
+}
+
+static int cgroup_release_agent_write(struct cgroup *cgrp, struct cftype *cft,
+          const char *buffer)
+{
+ BUILD_BUG_ON(sizeof(cgrp->root->release_agent_path) < PATH_MAX);
+ if (!cgroup_lock_live_group(cgrp))
+  return -ENODEV;
+ strcpy(cgrp->root->release_agent_path, buffer);
+ mutex_unlock(&cgroup_mutex);
+ return 0;
+}
+
+static int cgroup_release_agent_show(struct cgroup *cgrp, struct cftype *cft,
+          struct seq_file *seq)
+{
+ if (!cgroup_lock_live_group(cgrp))
+  return -ENODEV;
+ seq_puts(seq, cgrp->root->release_agent_path);
+ seq_putc(seq, '\n');
+ mutex_unlock(&cgroup_mutex);
+ return 0;
+}
+
 static ssize_t cgroup_write_X64(struct cgroup *cgrp, struct cftype *cft,
      struct file *file,
      const char __user *userbuf,
@@ -1443,10 +1478,6 @@ static ssize_t cgroup_common_file_write(
    else
     clear_bit(CGRP_NOTIFY_ON_RELEASE, &cgrp->flags);
    break;
- case FILE_RELEASE_AGENT:
-  BUILD_BUG_ON(sizeof(cgrp->root->release_agent_path) < PATH_MAX);
-  strcpy(cgrp->root->release_agent_path, buffer);
-  break;
   default:
    retval = -EINVAL;
    goto out2;
@@ -1506,49 +1537,6 @@ static ssize_t cgroup_read_s64(struct cg
  return simple_read_from_buffer(buf, nbytes, ppos, tmp, len);
 }
```

```
-static ssize_t cgroup_common_file_read(struct cgroup *cgrp,
-       struct cftype *cft,
-       struct file *file,
-       char __user *buf,
-       size_t nbytes, loff_t *ppos)
-{
- enum cgroup_filetype type = cft->private;
- char *page;
- ssize_t retval = 0;
- char *s;
-
- if (!(page = (char *)__get_free_page(GFP_KERNEL)))
-  return -ENOMEM;
-
- s = page;
-
- switch (type) {
- case FILE_RELEASE_AGENT:
- {
-  struct cgroupfs_root *root;
-  size_t n;
-  mutex_lock(&cgroup_mutex);
-  root = cgrp->root;
-  n = strnlen(root->release_agent_path,
-       sizeof(root->release_agent_path));
-  n = min(n, (size_t) PAGE_SIZE);
-  strncpy(s, root->release_agent_path, n);
-  mutex_unlock(&cgroup_mutex);
-  s += n;
-  break;
- }
- default:
-  retval = -EINVAL;
-  goto out;
- }
- *s++ = '\n';
-
- retval = simple_read_from_buffer(buf, nbytes, ppos, page, s - page);
-out:
- free_page((unsigned long)page);
- return retval;
-}
-
 static ssize_t cgroup_file_read(struct file *file, char __user *buf,
        size_t nbytes, loff_t *ppos)
 {
@@ -1606,6 +1594,7 @@ static int cgroup_seqfile_release(struct
```

```
 static struct file_operations cgroup_seqfile_operations = {
  .read = seq_read,
+ .write = cgroup_file_write,
  .llseek = seq_lseek,
  .release = cgroup_seqfile_release,
 };
@@ -2283,8 +2272,9 @@ static struct cftype files[] = {

 static struct cftype cft_release_agent = {
  .name = "release_agent",
- .read = cgroup_common_file_read,
- .write = cgroup_common_file_write,
+ .read_seq_string = cgroup_release_agent_show,
+ .write_string = cgroup_release_agent_write,
+ .max_write_len = PATH_MAX,
  .private = FILE_RELEASE_AGENT,
 };

@@ -3113,27 +3103,24 @@ static void cgroup_release_agent(struct
  while (!list_empty(&release_list)) {
   char *argv[3], *envp[3];
   int i;
-  char *pathbuf;
+  char *pathbuf = NULL, *agentbuf = NULL;
   struct cgroup *cgrp = list_entry(release_list.next,
         struct cgroup,
         release_list);
   list_del_init(&cgrp->release_list);
   spin_unlock(&release_list_lock);
   pathbuf = kmalloc(PAGE_SIZE, GFP_KERNEL);
-  if (!pathbuf) {
-   spin_lock(&release_list_lock);
-   continue;
-  }
-
-  if (cgroup_path(cgrp, pathbuf, PAGE_SIZE) < 0) {
-   kfree(pathbuf);
-   spin_lock(&release_list_lock);
-   continue;
-  }
+  if (!pathbuf)
+   goto continue_free;
+  if (cgroup_path(cgrp, pathbuf, PAGE_SIZE) < 0)
+   goto continue_free;
+  agentbuf = kstrdup(cgrp->root->release_agent_path, GFP_KERNEL);
+  if (!agentbuf)
+   goto continue_free;
```

```
   i = 0;
-  argv[i++] = cgrp->root->release_agent_path;
-  argv[i++] = (char *)pathbuf;
+  argv[i++] = agentbuf;
+  argv[i++] = pathbuf;
   argv[i] = NULL;

   i = 0;
@@ -3147,8 +3134,10 @@ static void cgroup_release_agent(struct
    * be a slow process */
   mutex_unlock(&cgroup_mutex);
   call_usermodehelper(argv[0], argv, envp, UMH_WAIT_EXEC);
-  kfree(pathbuf);
   mutex_lock(&cgroup_mutex);
+ continue_free:
+  kfree(pathbuf);
+  kfree(agentbuf);
   spin_lock(&release_list_lock);
  }
  spin_unlock(&release_list_lock);
Index: cws-2.6.26-rc5-mm3/include/linux/cgroup.h
===================================================================
--- cws-2.6.26-rc5-mm3.orig/include/linux/cgroup.h
+++ cws-2.6.26-rc5-mm3/include/linux/cgroup.h
@@ -295,6 +295,8 @@ int cgroup_add_files(struct cgroup *cgrp

 int cgroup_is_removed(const struct cgroup *cgrp);

+int cgroup_lock_live_group(struct cgroup *cgrp);
+
 int cgroup_path(const struct cgroup *cgrp, char *buf, int buflen);

 int cgroup_task_count(const struct cgroup *cgrp);

--
```


Subject: [PATCH 5/8] CGroup Files: Turn attach_task_by_pid directly into a cgroup write handler

Posted by Paul Menage on Fri, 20 Jun 2008 23:44:03 GMT

View Forum Message <> Reply to Message

This patch changes attach_task_by_pid() to take a u64 rather than a string; as a result it can be called directly as a control groups

write_u64 handler, and cgroup_common_file_write() can be removed.

Signed-off-by: Paul Menage <menage@google.com>

```
---
 kernel/cgroup.c |   80 +++++++++---------------------------------------------
 1 file changed, 14 insertions(+), 66 deletions(-)

Index: cws-2.6.26-rc5-mm3/kernel/cgroup.c
===================================================================
--- cws-2.6.26-rc5-mm3.orig/kernel/cgroup.c
+++ cws-2.6.26-rc5-mm3/kernel/cgroup.c
@@ -504,10 +504,6 @@ static struct css_set *find_css_set(
  * knows that the cgroup won't be removed, as cgroup_rmdir()
  * needs that mutex.
  *
- * The cgroup_common_file_write handler for operations that modify
- * the cgroup hierarchy holds cgroup_mutex across the entire operation,
- * single threading all such cgroup modifications across the system.
- *
  * The fork and exit callbacks cgroup_fork() and cgroup_exit(), don't
  * (usually) take cgroup_mutex.  These are the two most performance
  * critical pieces of code here.  The exception occurs on cgroup_exit(),
@@ -1279,18 +1275,14 @@ int cgroup_attach_task(struct cgroup *cg
 }

 /*
- * Attach task with pid 'pid' to cgroup 'cgrp'. Call with
- * cgroup_mutex, may take task_lock of task
+ * Attach task with pid 'pid' to cgroup 'cgrp'. Call with cgroup_mutex
+ * held. May take task_lock of task
  */
-static int attach_task_by_pid(struct cgroup *cgrp, char *pidbuf)
+static int attach_task_by_pid(struct cgroup *cgrp, u64 pid)
 {
- pid_t pid;
  struct task_struct *tsk;
  int ret;

- if (sscanf(pidbuf, "%d", &pid) != 1)
-  return -EIO;
-
  if (pid) {
   rcu_read_lock();
   tsk = find_task_by_vpid(pid);
@@ -1316,6 +1308,16 @@ static int attach_task_by_pid(struct cgr
  return ret;
 }
```

```
+static int cgroup_tasks_write(struct cgroup *cgrp, struct cftype *cft, u64 pid)
+{
+ int ret;
+ if (!cgroup_lock_live_group(cgrp))
+  return -ENODEV;
+ ret = attach_task_by_pid(cgrp, pid);
+ cgroup_unlock();
+ return ret;
+}
+
 /* The various types of files and directories in a cgroup file system */
 enum cgroup_filetype {
  FILE_ROOT,
@@ -1431,60 +1433,6 @@ static ssize_t cgroup_write_string(struc
  return retval;
 }

-static ssize_t cgroup_common_file_write(struct cgroup *cgrp,
-       struct cftype *cft,
-       struct file *file,
-       const char __user *userbuf,
-       size_t nbytes, loff_t *unused_ppos)
-{
- enum cgroup_filetype type = cft->private;
- char *buffer;
- int retval = 0;
-
- if (nbytes >= PATH_MAX)
-  return -E2BIG;
-
- /* +1 for nul-terminator */
- buffer = kmalloc(nbytes + 1, GFP_KERNEL);
- if (buffer == NULL)
-  return -ENOMEM;
-
- if (copy_from_user(buffer, userbuf, nbytes)) {
-  retval = -EFAULT;
-  goto out1;
- }
- buffer[nbytes] = 0; /* nul-terminate */
- strstrip(buffer); /* strip -just- trailing whitespace */
-
- mutex_lock(&cgroup_mutex);
-
- /*
-  * This was already checked for in cgroup_file_write(), but
-  * check again now we're holding cgroup_mutex.
```

```
-  */
- if (cgroup_is_removed(cgrp)) {
-  retval = -ENODEV;
-  goto out2;
- }
-
- switch (type) {
- case FILE_TASKLIST:
-  retval = attach_task_by_pid(cgrp, buffer);
-  break;
- default:
-  retval = -EINVAL;
-  goto out2;
- }
-
- if (retval == 0)
-  retval = nbytes;
-out2:
- mutex_unlock(&cgroup_mutex);
-out1:
- kfree(buffer);
- return retval;
-}
-
 static ssize_t cgroup_file_write(struct file *file, const char __user *buf,
       size_t nbytes, loff_t *ppos)
 {
@@ -2262,7 +2210,7 @@ static struct cftype files[] = {
   .name = "tasks",
   .open = cgroup_tasks_open,
   .read = cgroup_tasks_read,
-  .write = cgroup_common_file_write,
+  .write_u64 = cgroup_tasks_write,
   .release = cgroup_tasks_release,
   .private = FILE_TASKLIST,
  },

--
```

Subject: [PATCH 7/8] CGroup Files: Convert devcgroup_access_write() into a cgroup write_string() handler
Posted by Paul Menage on Fri, 20 Jun 2008 23:44:05 GMT

This patch converts devcgroup_access_write() from a raw file handler
into a handler for the cgroup write_string() method. This allows some
boilerplate copying/locking/checking to be removed and simplifies the
cleanup path, since these functions are performed by the cgroups
framework before calling the handler.

Signed-off-by: Paul Menage <menage@google.com>

```
---
 security/device_cgroup.c |  103 ++++++++++++++++-----------------------------
 1 file changed, 39 insertions(+), 64 deletions(-)

Index: cws-2.6.26-rc5-mm3/security/device_cgroup.c
===================================================================
--- cws-2.6.26-rc5-mm3.orig/security/device_cgroup.c
+++ cws-2.6.26-rc5-mm3/security/device_cgroup.c
@@ -59,6 +59,11 @@ static inline struct dev_cgroup *cgroup_
  return css_to_devcgroup(cgroup_subsys_state(cgroup, devices_subsys_id));
 }

+static inline struct dev_cgroup *task_devcgroup(struct task_struct *task)
+{
+ return css_to_devcgroup(task_subsys_state(task, devices_subsys_id));
+}
+
 struct cgroup_subsys devices_subsys;

 static int devcgroup_can_attach(struct cgroup_subsys *ss,
@@ -312,10 +317,10 @@ static int may_access_whitelist(struct d
  * when adding a new allow rule to a device whitelist, the rule
  * must be allowed in the parent device
  */
-static int parent_has_perm(struct cgroup *childcg,
+static int parent_has_perm(struct dev_cgroup *childcg,
       struct dev_whitelist_item *wh)
 {
- struct cgroup *pcg = childcg->parent;
+ struct cgroup *pcg = childcg->css.cgroup->parent;
  struct dev_cgroup *parent;
  int ret;

@@ -341,39 +346,18 @@ static int parent_has_perm(struct cgroup
  * new access is only allowed if you're in the top-level cgroup, or your
  * parent cgroup has the access you're asking for.
  */
-static ssize_t devcgroup_access_write(struct cgroup *cgroup, struct cftype *cft,
-    struct file *file, const char __user *userbuf,
```

```
-    size_t nbytes, loff_t *ppos)
-{
- struct cgroup *cur_cgroup;
- struct dev_cgroup *devcgroup, *cur_devcgroup;
- int filetype = cft->private;
- char *buffer, *b;
+static int devcgroup_update_access(struct dev_cgroup *devcgroup,
+       int filetype, const char *buffer)
+{
+ struct dev_cgroup *cur_devcgroup;
+ const char *b;
  int retval = 0, count;
  struct dev_whitelist_item wh;

  if (!capable(CAP_SYS_ADMIN))
   return -EPERM;

- devcgroup = cgroup_to_devcgroup(cgroup);
- cur_cgroup = task_cgroup(current, devices_subsys.subsys_id);
- cur_devcgroup = cgroup_to_devcgroup(cur_cgroup);
-
- buffer = kmalloc(nbytes+1, GFP_KERNEL);
- if (!buffer)
-  return -ENOMEM;
-
- if (copy_from_user(buffer, userbuf, nbytes)) {
-  retval = -EFAULT;
-  goto out1;
- }
- buffer[nbytes] = 0; /* nul-terminate */
-
- cgroup_lock();
- if (cgroup_is_removed(cgroup)) {
-  retval = -ENODEV;
-  goto out2;
- }
+ cur_devcgroup = task_devcgroup(current);

  memset(&wh, 0, sizeof(wh));
  b = buffer;
@@ -390,14 +374,11 @@ static ssize_t devcgroup_access_write(st
   wh.type = DEV_CHAR;
   break;
  default:
-  retval = -EINVAL;
-  goto out2;
+  return -EINVAL;
  }
```

```
  b++;
- if (!isspace(*b)) {
-  retval = -EINVAL;
-  goto out2;
- }
+ if (!isspace(*b))
+  return -EINVAL;
  b++;
  if (*b == '*') {
   wh.major = ~0;
@@ -409,13 +390,10 @@ static ssize_t devcgroup_access_write(st
    b++;
   }
  } else {
-  retval = -EINVAL;
-  goto out2;
- }
- if (*b != ':') {
-  retval = -EINVAL;
-  goto out2;
+  return -EINVAL;
  }
+ if (*b != ':')
+  return -EINVAL;
  b++;

  /* read minor */
@@ -429,13 +407,10 @@ static ssize_t devcgroup_access_write(st
    b++;
   }
  } else {
-  retval = -EINVAL;
-  goto out2;
- }
- if (!isspace(*b)) {
-  retval = -EINVAL;
-  goto out2;
+  return -EINVAL;
  }
+ if (!isspace(*b))
+  return -EINVAL;
  for (b++, count = 0; count < 3; count++, b++) {
   switch (*b) {
   case 'r':
@@ -452,8 +427,7 @@ static ssize_t devcgroup_access_write(st
    count = 3;
    break;
   default:
```

```
-   retval = -EINVAL;
-   goto out2;
+   return -EINVAL;
   }
  }

@@ -461,38 +435,39 @@ handle:
  retval = 0;
  switch (filetype) {
  case DEVCG_ALLOW:
-  if (!parent_has_perm(cgroup, &wh))
-   retval = -EPERM;
-  else
-   retval = dev_whitelist_add(devcgroup, &wh);
-  break;
+  if (!parent_has_perm(devcgroup, &wh))
+   return -EPERM;
+  return dev_whitelist_add(devcgroup, &wh);
  case DEVCG_DENY:
   dev_whitelist_rm(devcgroup, &wh);
   break;
  default:
-  retval = -EINVAL;
-  goto out2;
+  return -EINVAL;
  }
+ return 0;
+}

- if (retval == 0)
-  retval = nbytes;
-
-out2:
+static int devcgroup_access_write(struct cgroup *cgrp, struct cftype *cft,
+       const char *buffer)
+{
+ int retval;
+ if (!cgroup_lock_live_group(cgrp))
+  return -ENODEV;
+ retval = devcgroup_update_access(cgroup_to_devcgroup(cgrp),
+       cft->private, buffer);
  cgroup_unlock();
-out1:
- kfree(buffer);
  return retval;
 }

 static struct cftype dev_cgroup_files[] = {
```

```
 {
  .name = "allow",
-  .write  = devcgroup_access_write,
+  .write_string  = devcgroup_access_write,
  .private = DEVCG_ALLOW,
 },
 {
  .name = "deny",
-  .write = devcgroup_access_write,
+  .write_string = devcgroup_access_write,
  .private = DEVCG_DENY,
 },
 {

--
```

Subject: Re: [PATCH 2/8] CGroup Files: Add write_string cgroup control file method
Posted by Balbir Singh on Sun, 22 Jun 2008 14:32:36 GMT
View Forum Message <> Reply to Message

* menage@google.com <menage@google.com> [2008-06-20 16:44:00]:

> This patch adds a write_string() method for cgroups control files. The
> semantics are that a buffer is copied from userspace to kernelspace
> and the handler function invoked on that buffer. The buffer is
> guaranteed to be nul-terminated, and no longer than max_write_len
> (defaulting to 64 bytes if unspecified). Later patches will convert
> existing raw file write handlers in control group subsystems to use
> this method.
>
> Signed-off-by: Paul Menage <menage@google.com>
>
> ---
> include/linux/cgroup.h | 14 +++++++++++++
> kernel/cgroup.c | 35 +++++++++++++++++++++++++++++++++++
> 2 files changed, 49 insertions(+)
>
> Index: cws-2.6.26-rc5-mm3/include/linux/cgroup.h
> ===
> --- cws-2.6.26-rc5-mm3.orig/include/linux/cgroup.h
> +++ cws-2.6.26-rc5-mm3/include/linux/cgroup.h
> @@ -205,6 +205,13 @@ struct cftype {
>   * subsystem, followed by a period */

>   char name[MAX_CFTYPE_NAME];
>   int private;
> +
> + /*
> + * If non-zero, defines the maximum length of string that can
> + * be passed to write_string; defaults to 64
> + */
> + size_t max_write_len;
> +
>   int (*open)(struct inode *inode, struct file *file);
>   ssize_t (*read)(struct cgroup *cgrp, struct cftype *cft,
>   struct file *file,
> @@ -249,6 +256,13 @@ struct cftype {
>   int (*write_s64)(struct cgroup *cgrp, struct cftype *cft, s64 val);
>
>   /*
> + * write_string() is passed a nul-terminated kernelspace
> + * buffer of maximum length determined by max_write_len.
> + * Returns 0 or -ve error code.
> + */
> + int (*write_string)(struct cgroup *cgrp, struct cftype *cft,
> +   const char *buffer);
> + /*
>   * trigger() callback can be used to get some kick from the
>   * userspace, when the actual string written is not important
>   * at all. The private field can be used to determine the
> Index: cws-2.6.26-rc5-mm3/kernel/cgroup.c
> ===
> --- cws-2.6.26-rc5-mm3.orig/kernel/cgroup.c
> +++ cws-2.6.26-rc5-mm3/kernel/cgroup.c
> @@ -1363,6 +1363,39 @@ static ssize_t cgroup_write_X64(struct c
>   return retval;
> }
>
> +static ssize_t cgroup_write_string(struct cgroup *cgrp, struct cftype *cft,
> +   struct file *file,
> +   const char __user *userbuf,
> +   size_t nbytes, loff_t *unused_ppos)
> +{
> + char local_buffer[64];

64? a define would be more meaningful

> + int retval = 0;
> + size_t max_bytes = cft->max_write_len;
> + char *buffer = local_buffer;
> +
> + if (!max_bytes)

> + max_bytes = sizeof(local_buffer) - 1;
> + if (nbytes >= max_bytes)
> + return -E2BIG;
> + /* Allocate a dynamic buffer if we need one */
> + if (nbytes >= sizeof(local_buffer)) {
> + buffer = kmalloc(nbytes + 1, GFP_KERNEL);
> + if (buffer == NULL)
> +   return -ENOMEM;
> + }
> + if (nbytes && copy_from_user(buffer, userbuf, nbytes))
> + return -EFAULT;
> +
> + buffer[nbytes] = 0;   /* nul-terminate */
> + strstrip(buffer);
> + retval = cft->write_string(cgrp, cft, buffer);
> + if (!retval)
> + retval = nbytes;
> + if (buffer != local_buffer)
> + kfree(buffer);
> + return retval;
> +}
> +
> static ssize_t cgroup_common_file_write(struct cgroup *cgrp,
>   struct cftype *cft,
>   struct file *file,
> @@ -1440,6 +1473,8 @@ static ssize_t cgroup_file_write(struct
>   return cft->write(cgrp, cft, file, buf, nbytes, ppos);
>   if (cft->write_u64 || cft->write_s64)
>   return cgroup_write_X64(cgrp, cft, file, buf, nbytes, ppos);
> + if (cft->write_string)
> + return cgroup_write_string(cgrp, cft, file, buf, nbytes, ppos);
>   if (cft->trigger) {
>   int ret = cft->trigger(cgrp, (unsigned int)cft->private);
>   return ret ? ret : nbytes;
>
> --

Looks good

Acked-by: Balbir Singh <balbir@linux.vnet.ibm.com>

--
 Warm Regards,
 Balbir Singh
 Linux Technology Center
 IBM, ISTL

Subject: Re: [PATCH 2/8] CGroup Files: Add write_string cgroup control file method
Posted by Paul Menage on Tue, 24 Jun 2008 14:27:14 GMT
View Forum Message <> Reply to Message

On Sun, Jun 22, 2008 at 7:32 AM, Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
>> +static ssize_t cgroup_write_string(struct cgroup *cgrp, struct cftype *cft,
>> +   struct file *file,
>> +   const char __user *userbuf,
>> +   size_t nbytes, loff_t *unused_ppos)
>> +{
>> +   char local_buffer[64];
>
> 64? a define would be more meaningful

Potentially, although it would be unlikely to be reused anywhere else.
It's just meant to be a size that's big enough for any numerical
value, and for the vast majority of other writes.

Paul

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

Subject: Re: [PATCH 2/8] CGroup Files: Add write_string cgroup control file method
Posted by serue on Tue, 24 Jun 2008 15:34:34 GMT
View Forum Message <> Reply to Message

Quoting menage@google.com (menage@google.com):
> This patch adds a write_string() method for cgroups control files. The
> semantics are that a buffer is copied from userspace to kernelspace
> and the handler function invoked on that buffer. The buffer is
> guaranteed to be nul-terminated, and no longer than max_write_len
> (defaulting to 64 bytes if unspecified). Later patches will convert
> existing raw file write handlers in control group subsystems to use
> this method.
>
> Signed-off-by: Paul Menage <menage@google.com>

Looks sane to me.

Acked-by: Serge Hallyn <serue@us.ibm.com>

thanks,
-serge

>
> ---
> include/linux/cgroup.h | 14 +++++++++++++
> kernel/cgroup.c | 35 +++++++++++++++++++++++++++++++++++
> 2 files changed, 49 insertions(+)
>
> Index: cws-2.6.26-rc5-mm3/include/linux/cgroup.h
> ===
> --- cws-2.6.26-rc5-mm3.orig/include/linux/cgroup.h
> +++ cws-2.6.26-rc5-mm3/include/linux/cgroup.h
> @@ -205,6 +205,13 @@ struct cftype {
>   * subsystem, followed by a period */
>   char name[MAX_CFTYPE_NAME];
>   int private;
> +
> + /*
> + * If non-zero, defines the maximum length of string that can
> + * be passed to write_string; defaults to 64
> + */
> + size_t max_write_len;
> +
>   int (*open)(struct inode *inode, struct file *file);
>   ssize_t (*read)(struct cgroup *cgrp, struct cftype *cft,
>   struct file *file,
> @@ -249,6 +256,13 @@ struct cftype {
>   int (*write_s64)(struct cgroup *cgrp, struct cftype *cft, s64 val);
>
>   /*
> + * write_string() is passed a nul-terminated kernelspace
> + * buffer of maximum length determined by max_write_len.
> + * Returns 0 or -ve error code.
> + */
> + int (*write_string)(struct cgroup *cgrp, struct cftype *cft,
> +   const char *buffer);
> + /*
>   * trigger() callback can be used to get some kick from the
>   * userspace, when the actual string written is not important
>   * at all. The private field can be used to determine the
> Index: cws-2.6.26-rc5-mm3/kernel/cgroup.c
> ===
> --- cws-2.6.26-rc5-mm3.orig/kernel/cgroup.c
> +++ cws-2.6.26-rc5-mm3/kernel/cgroup.c
> @@ -1363,6 +1363,39 @@ static ssize_t cgroup_write_X64(struct c

```
>   return retval;
>  }
>
> +static ssize_t cgroup_write_string(struct cgroup *cgrp, struct cftype *cft,
> +       struct file *file,
> +       const char __user *userbuf,
> +       size_t nbytes, loff_t *unused_ppos)
> +{
> + char local_buffer[64];
> + int retval = 0;
> + size_t max_bytes = cft->max_write_len;
> + char *buffer = local_buffer;
> +
> + if (!max_bytes)
> +  max_bytes = sizeof(local_buffer) - 1;
> + if (nbytes >= max_bytes)
> +  return -E2BIG;
> + /* Allocate a dynamic buffer if we need one */
> + if (nbytes >= sizeof(local_buffer)) {
> +  buffer = kmalloc(nbytes + 1, GFP_KERNEL);
> +  if (buffer == NULL)
> +   return -ENOMEM;
> + }
> + if (nbytes && copy_from_user(buffer, userbuf, nbytes))
> +  return -EFAULT;
> +
> + buffer[nbytes] = 0;     /* nul-terminate */
> + strstrip(buffer);
> + retval = cft->write_string(cgrp, cft, buffer);
> + if (!retval)
> +  retval = nbytes;
> + if (buffer != local_buffer)
> +  kfree(buffer);
> + return retval;
> +}
> +
>  static ssize_t cgroup_common_file_write(struct cgroup *cgrp,
>         struct cftype *cft,
>         struct file *file,
> @@ -1440,6 +1473,8 @@ static ssize_t cgroup_file_write(struct
>    return cft->write(cgrp, cft, file, buf, nbytes, ppos);
>   if (cft->write_u64 || cft->write_s64)
>    return cgroup_write_X64(cgrp, cft, file, buf, nbytes, ppos);
> + if (cft->write_string)
> +  return cgroup_write_string(cgrp, cft, file, buf, nbytes, ppos);
>   if (cft->trigger) {
>    int ret = cft->trigger(cgrp, (unsigned int)cft->private);
>    return ret ? ret : nbytes;
```

>
> --

_____

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

---

Subject: Re: [PATCH 3/8] CGroup Files: Move the release_agent file to use typed handlers
Posted by serue on Tue, 24 Jun 2008 15:56:34 GMT
View Forum Message <> Reply to Message

Quoting menage@google.com (menage@google.com):
> Adds cgroup_release_agent_write() and cgroup_release_agent_show()
> methods to handle writing/reading the path to a cgroup hierarchy's
> release agent. As a result, cgroup_common_file_read() is now unnecessary.
>
> As part of the change, a previously-tolerated race in
> cgroup_release_agent() is avoided by copying the current
> release_agent_path prior to calling call_usermode_helper().
>
> Signed-off-by: Paul Menage <menage@google.com>
>
> ---
> include/linux/cgroup.h | 2
> kernel/cgroup.c | 125 ++++++++++++++++++++++++++++++++----------------
> 2 files changed, 59 insertions(+), 68 deletions(-)
>
> Index: cws-2.6.26-rc5-mm3/kernel/cgroup.c
> ===
> --- cws-2.6.26-rc5-mm3.orig/kernel/cgroup.c
> +++ cws-2.6.26-rc5-mm3/kernel/cgroup.c
> @@ -89,11 +89,7 @@ struct cgroupfs_root {
>   /* Hierarchy-specific flags */
>   unsigned long flags;
>
> - /* The path to use for release notifications. No locking
> - * between setting and use - so if userspace updates this
> - * while child cgroups exist, you could miss a
> - * notification. We ensure that it's always a valid
> - * NUL-terminated string */
> + /* The path to use for release notifications. */
>   char release_agent_path[PATH_MAX];
> };
>
> @@ -1329,6 +1325,45 @@ enum cgroup_filetype {
>   FILE_RELEASE_AGENT,

> };
>
> +/**
> + * cgroup_lock_live_group - take cgroup_mutex and check that cgrp is alive.
> + * @cgrp: the cgroup to be checked for liveness
> + *
> + * Returns true (with lock held) on success, or false (with no lock
> + * held) on failure.
> + */
> +int cgroup_lock_live_group(struct cgroup *cgrp)

Would seem more consistent to call the return type bool, but otherwise
this patch looks good.

Acked-by: Serge Hallyn <serue@us.ibm.com>

thanks,
-serge

> +{
> + mutex_lock(&cgroup_mutex);
> + if (cgroup_is_removed(cgrp)) {
> + mutex_unlock(&cgroup_mutex);
> + return false;
> + }
> + return true;
> +}
> +
> +static int cgroup_release_agent_write(struct cgroup *cgrp, struct cftype *cft,
> +   const char *buffer)
> +{
> + BUILD_BUG_ON(sizeof(cgrp->root->release_agent_path) < PATH_MAX);
> + if (!cgroup_lock_live_group(cgrp))
> + return -ENODEV;
> + strcpy(cgrp->root->release_agent_path, buffer);
> + mutex_unlock(&cgroup_mutex);
> + return 0;
> +}
> +
> +static int cgroup_release_agent_show(struct cgroup *cgrp, struct cftype *cft,
> +   struct seq_file *seq)
> +{
> + if (!cgroup_lock_live_group(cgrp))
> + return -ENODEV;
> + seq_puts(seq, cgrp->root->release_agent_path);
> + seq_putc(seq, '\n');
> + mutex_unlock(&cgroup_mutex);
> + return 0;

```
> +}
> +
>  static ssize_t cgroup_write_X64(struct cgroup *cgrp, struct cftype *cft,
>      struct file *file,
>      const char __user *userbuf,
> @@ -1443,10 +1478,6 @@ static ssize_t cgroup_common_file_write(
>    else
>     clear_bit(CGRP_NOTIFY_ON_RELEASE, &cgrp->flags);
>    break;
> - case FILE_RELEASE_AGENT:
> -  BUILD_BUG_ON(sizeof(cgrp->root->release_agent_path) < PATH_MAX);
> -  strcpy(cgrp->root->release_agent_path, buffer);
> -  break;
>   default:
>    retval = -EINVAL;
>    goto out2;
> @@ -1506,49 +1537,6 @@ static ssize_t cgroup_read_s64(struct cg
>   return simple_read_from_buffer(buf, nbytes, ppos, tmp, len);
>  }
>
> -static ssize_t cgroup_common_file_read(struct cgroup *cgrp,
> -       struct cftype *cft,
> -       struct file *file,
> -       char __user *buf,
> -       size_t nbytes, loff_t *ppos)
> -{
> - enum cgroup_filetype type = cft->private;
> - char *page;
> - ssize_t retval = 0;
> - char *s;
> -
> - if (!(page = (char *)__get_free_page(GFP_KERNEL)))
> -  return -ENOMEM;
> -
> - s = page;
> -
> - switch (type) {
> - case FILE_RELEASE_AGENT:
> - {
> -  struct cgroupfs_root *root;
> -  size_t n;
> -  mutex_lock(&cgroup_mutex);
> -  root = cgrp->root;
> -  n = strnlen(root->release_agent_path,
> -       sizeof(root->release_agent_path));
> -  n = min(n, (size_t) PAGE_SIZE);
> -  strncpy(s, root->release_agent_path, n);
> -  mutex_unlock(&cgroup_mutex);
```

```
> -  s += n;
> -  break;
> - }
> - default:
> -  retval = -EINVAL;
> -  goto out;
> - }
> - *s++ = '\n';
> -
> - retval = simple_read_from_buffer(buf, nbytes, ppos, page, s - page);
> -out:
> - free_page((unsigned long)page);
> - return retval;
> -}
> -
>  static ssize_t cgroup_file_read(struct file *file, char __user *buf,
>         size_t nbytes, loff_t *ppos)
>  {
> @@ -1606,6 +1594,7 @@ static int cgroup_seqfile_release(struct
>
>  static struct file_operations cgroup_seqfile_operations = {
>   .read = seq_read,
> + .write = cgroup_file_write,
>   .llseek = seq_lseek,
>   .release = cgroup_seqfile_release,
>  };
> @@ -2283,8 +2272,9 @@ static struct cftype files[] = {
>
>  static struct cftype cft_release_agent = {
>   .name = "release_agent",
> - .read = cgroup_common_file_read,
> - .write = cgroup_common_file_write,
> + .read_seq_string = cgroup_release_agent_show,
> + .write_string = cgroup_release_agent_write,
> + .max_write_len = PATH_MAX,
>   .private = FILE_RELEASE_AGENT,
>  };
>
> @@ -3113,27 +3103,24 @@ static void cgroup_release_agent(struct
>   while (!list_empty(&release_list)) {
>    char *argv[3], *envp[3];
>    int i;
> -  char *pathbuf;
> +  char *pathbuf = NULL, *agentbuf = NULL;
>    struct cgroup *cgrp = list_entry(release_list.next,
>           struct cgroup,
>           release_list);
>    list_del_init(&cgrp->release_list);
```

```
>    spin_unlock(&release_list_lock);
>    pathbuf = kmalloc(PAGE_SIZE, GFP_KERNEL);
> -  if (!pathbuf) {
> -   spin_lock(&release_list_lock);
> -   continue;
> -  }
> -
> -  if (cgroup_path(cgrp, pathbuf, PAGE_SIZE) < 0) {
> -   kfree(pathbuf);
> -   spin_lock(&release_list_lock);
> -   continue;
> -  }
> +  if (!pathbuf)
> +   goto continue_free;
> +  if (cgroup_path(cgrp, pathbuf, PAGE_SIZE) < 0)
> +   goto continue_free;
> +  agentbuf = kstrdup(cgrp->root->release_agent_path, GFP_KERNEL);
> +  if (!agentbuf)
> +   goto continue_free;
>
>    i = 0;
> -  argv[i++] = cgrp->root->release_agent_path;
> -  argv[i++] = (char *)pathbuf;
> +  argv[i++] = agentbuf;
> +  argv[i++] = pathbuf;
>    argv[i] = NULL;
>
>    i = 0;
> @@ -3147,8 +3134,10 @@ static void cgroup_release_agent(struct
>     * be a slow process */
>    mutex_unlock(&cgroup_mutex);
>    call_usermodehelper(argv[0], argv, envp, UMH_WAIT_EXEC);
> -  kfree(pathbuf);
>    mutex_lock(&cgroup_mutex);
> + continue_free:
> +  kfree(pathbuf);
> +  kfree(agentbuf);
>    spin_lock(&release_list_lock);
>   }
>   spin_unlock(&release_list_lock);
> Index: cws-2.6.26-rc5-mm3/include/linux/cgroup.h
> ===================================================================
> --- cws-2.6.26-rc5-mm3.orig/include/linux/cgroup.h
> +++ cws-2.6.26-rc5-mm3/include/linux/cgroup.h
> @@ -295,6 +295,8 @@ int cgroup_add_files(struct cgroup *cgrp
>
>  int cgroup_is_removed(const struct cgroup *cgrp);
>
```

> +int cgroup_lock_live_group(struct cgroup *cgrp);
> +
> int cgroup_path(const struct cgroup *cgrp, char *buf, int buflen);
>
> int cgroup_task_count(const struct cgroup *cgrp);
>
> --

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

---

Subject: Re: [PATCH 7/8] CGroup Files: Convert devcgroup_access_write() into a cgroup write_string() handler
Posted by serue on Tue, 24 Jun 2008 16:21:23 GMT
View Forum Message <> Reply to Message

Quoting menage@google.com (menage@google.com):
> This patch converts devcgroup_access_write() from a raw file handler
> into a handler for the cgroup write_string() method. This allows some
> boilerplate copying/locking/checking to be removed and simplifies the
> cleanup path, since these functions are performed by the cgroups
> framework before calling the handler.
>
> Signed-off-by: Paul Menage <menage@google.com>

Looks good. I'll have to test later.

Acked-by: Serge Hallyn <serue@us.ibm.com>

thanks,
-serge

>
> ---
> security/device_cgroup.c | 103 ++++++++++++++++++++++++++++++----------------
> 1 file changed, 39 insertions(+), 64 deletions(-)
>
> Index: cws-2.6.26-rc5-mm3/security/device_cgroup.c
> ===
> --- cws-2.6.26-rc5-mm3.orig/security/device_cgroup.c
> +++ cws-2.6.26-rc5-mm3/security/device_cgroup.c
> @@ -59,6 +59,11 @@ static inline struct dev_cgroup *cgroup_
>   return css_to_devcgroup(cgroup_subsys_state(cgroup, devices_subsys_id));
> }
>
> +static inline struct dev_cgroup *task_devcgroup(struct task_struct *task)

> +{
> + return css_to_devcgroup(task_subsys_state(task, devices_subsys_id));
> +}
> +
> struct cgroup_subsys devices_subsys;
>
> static int devcgroup_can_attach(struct cgroup_subsys *ss,
> @@ -312,10 +317,10 @@ static int may_access_whitelist(struct d
>   * when adding a new allow rule to a device whitelist, the rule
>   * must be allowed in the parent device
>   */
> -static int parent_has_perm(struct cgroup *childcg,
> +static int parent_has_perm(struct dev_cgroup *childcg,
>   struct dev_whitelist_item *wh)
> {
> - struct cgroup *pcg = childcg->parent;
> + struct cgroup *pcg = childcg->css.cgroup->parent;
>   struct dev_cgroup *parent;
>   int ret;
>
> @@ -341,39 +346,18 @@ static int parent_has_perm(struct cgroup
>   * new access is only allowed if you're in the top-level cgroup, or your
>   * parent cgroup has the access you're asking for.
>   */
> -static ssize_t devcgroup_access_write(struct cgroup *cgroup, struct cftype *cft,
> -   struct file *file, const char __user *userbuf,
> -   size_t nbytes, loff_t *ppos)
> -{
> - struct cgroup *cur_cgroup;
> - struct dev_cgroup *devcgroup, *cur_devcgroup;
> - int filetype = cft->private;
> - char *buffer, *b;
> +static int devcgroup_update_access(struct dev_cgroup *devcgroup,
> +   int filetype, const char *buffer)
> +{
> + struct dev_cgroup *cur_devcgroup;
> + const char *b;
>   int retval = 0, count;
>   struct dev_whitelist_item wh;
>
>   if (!capable(CAP_SYS_ADMIN))
>   return -EPERM;
>
> - devcgroup = cgroup_to_devcgroup(cgroup);
> - cur_cgroup = task_cgroup(current, devices_subsys.subsys_id);
> - cur_devcgroup = cgroup_to_devcgroup(cur_cgroup);
> -
> - buffer = kmalloc(nbytes+1, GFP_KERNEL);

```
> - if (!buffer)
> -  return -ENOMEM;
> -
> - if (copy_from_user(buffer, userbuf, nbytes)) {
> -  retval = -EFAULT;
> -  goto out1;
> - }
> - buffer[nbytes] = 0; /* nul-terminate */
> -
> - cgroup_lock();
> - if (cgroup_is_removed(cgroup)) {
> -  retval = -ENODEV;
> -  goto out2;
> - }
> + cur_devcgroup = task_devcgroup(current);
>
>   memset(&wh, 0, sizeof(wh));
>   b = buffer;
> @@ -390,14 +374,11 @@ static ssize_t devcgroup_access_write(st
>    wh.type = DEV_CHAR;
>    break;
>   default:
> -  retval = -EINVAL;
> -  goto out2;
> +  return -EINVAL;
>   }
>   b++;
> - if (!isspace(*b)) {
> -  retval = -EINVAL;
> -  goto out2;
> - }
> + if (!isspace(*b))
> +  return -EINVAL;
>   b++;
>   if (*b == '*') {
>    wh.major = ~0;
> @@ -409,13 +390,10 @@ static ssize_t devcgroup_access_write(st
>     b++;
>    }
>   } else {
> -  retval = -EINVAL;
> -  goto out2;
> - }
> - if (*b != ':') {
> -  retval = -EINVAL;
> -  goto out2;
> +  return -EINVAL;
>   }
```

```
> + if (*b != ':')
> +  return -EINVAL;
>    b++;
>
>    /* read minor */
> @@ -429,13 +407,10 @@ static ssize_t devcgroup_access_write(st
>      b++;
>     }
>    } else {
> -  retval = -EINVAL;
> -  goto out2;
> - }
> - if (!isspace(*b)) {
> -  retval = -EINVAL;
> -  goto out2;
> +  return -EINVAL;
>    }
> + if (!isspace(*b))
> +  return -EINVAL;
>    for (b++, count = 0; count < 3; count++, b++) {
>     switch (*b) {
>     case 'r':
> @@ -452,8 +427,7 @@ static ssize_t devcgroup_access_write(st
>      count = 3;
>      break;
>     default:
> -   retval = -EINVAL;
> -   goto out2;
> +   return -EINVAL;
>     }
>    }
>
> @@ -461,38 +435,39 @@ handle:
>    retval = 0;
>    switch (filetype) {
>    case DEVCG_ALLOW:
> -  if (!parent_has_perm(cgroup, &wh))
> -   retval = -EPERM;
> -  else
> -   retval = dev_whitelist_add(devcgroup, &wh);
> -  break;
> +  if (!parent_has_perm(devcgroup, &wh))
> +   return -EPERM;
> +  return dev_whitelist_add(devcgroup, &wh);
>    case DEVCG_DENY:
>     dev_whitelist_rm(devcgroup, &wh);
>     break;
>    default:
```

```
> -  retval = -EINVAL;
> -  goto out2;
> +  return -EINVAL;
>   }
> + return 0;
> +}
>
> - if (retval == 0)
> -  retval = nbytes;
> -
> -out2:
> +static int devcgroup_access_write(struct cgroup *cgrp, struct cftype *cft,
> +       const char *buffer)
> +{
> + int retval;
> + if (!cgroup_lock_live_group(cgrp))
> +  return -ENODEV;
> + retval = devcgroup_update_access(cgroup_to_devcgroup(cgrp),
> +       cft->private, buffer);
>   cgroup_unlock();
> -out1:
> - kfree(buffer);
>   return retval;
>  }
>
>  static struct cftype dev_cgroup_files[] = {
>   {
>    .name = "allow",
> -  .write  = devcgroup_access_write,
> +  .write_string  = devcgroup_access_write,
>    .private = DEVCG_ALLOW,
>   },
>   {
>    .name = "deny",
> -  .write = devcgroup_access_write,
> +  .write_string = devcgroup_access_write,
>    .private = DEVCG_DENY,
>   },
>   {
>
> --
```

_____

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

Subject: Re: [PATCH 2/8] CGroup Files: Add write_string cgroup control file method
Posted by akpm on Tue, 24 Jun 2008 23:19:23 GMT
View Forum Message <> Reply to Message

On Fri, 20 Jun 2008 16:44:00 -0700
menage@google.com wrote:

> This patch adds a write_string() method for cgroups control files. The
> semantics are that a buffer is copied from userspace to kernelspace
> and the handler function invoked on that buffer. The buffer is
> guaranteed to be nul-terminated, and no longer than max_write_len
> (defaulting to 64 bytes if unspecified). Later patches will convert
> existing raw file write handlers in control group subsystems to use
> this method.
>
> Signed-off-by: Paul Menage <menage@google.com>
>
> ---
> include/linux/cgroup.h | 14 ++++++++++++
> kernel/cgroup.c | 35 +++++++++++++++++++++++++++++++++++
> 2 files changed, 49 insertions(+)
>
> Index: cws-2.6.26-rc5-mm3/include/linux/cgroup.h
> ===
> --- cws-2.6.26-rc5-mm3.orig/include/linux/cgroup.h
> +++ cws-2.6.26-rc5-mm3/include/linux/cgroup.h
> @@ -205,6 +205,13 @@ struct cftype {
>   * subsystem, followed by a period */
>   char name[MAX_CFTYPE_NAME];
>   int private;
> +
> + /*
> + * If non-zero, defines the maximum length of string that can
> + * be passed to write_string; defaults to 64
> + */
> + size_t max_write_len;
> +
>   int (*open)(struct inode *inode, struct file *file);
>   ssize_t (*read)(struct cgroup *cgrp, struct cftype *cft,
>   struct file *file,
> @@ -249,6 +256,13 @@ struct cftype {
>   int (*write_s64)(struct cgroup *cgrp, struct cftype *cft, s64 val);
>
>   /*
> + * write_string() is passed a nul-terminated kernelspace
> + * buffer of maximum length determined by max_write_len.
> + * Returns 0 or -ve error code.
> + */
> + int (*write_string)(struct cgroup *cgrp, struct cftype *cft,

> + const char *buffer);

Everything seems to use size_t (or ssize_t?) except for the ->write_string
return value. Can any of this be improved?

> + /*
> * trigger() callback can be used to get some kick from the
> * userspace, when the actual string written is not important
> * at all. The private field can be used to determine the
> Index: cws-2.6.26-rc5-mm3/kernel/cgroup.c
> ===
> --- cws-2.6.26-rc5-mm3.orig/kernel/cgroup.c
> +++ cws-2.6.26-rc5-mm3/kernel/cgroup.c
> @@ -1363,6 +1363,39 @@ static ssize_t cgroup_write_X64(struct c
> return retval;
> }
>
> +static ssize_t cgroup_write_string(struct cgroup *cgrp, struct cftype *cft,
> + struct file *file,
> + const char __user *userbuf,
> + size_t nbytes, loff_t *unused_ppos)
> +{
> + char local_buffer[64];
> + int retval = 0;
> + size_t max_bytes = cft->max_write_len;
> + char *buffer = local_buffer;
> +
> + if (!max_bytes)
> + max_bytes = sizeof(local_buffer) - 1;
> + if (nbytes >= max_bytes)
> + return -E2BIG;
> + /* Allocate a dynamic buffer if we need one */
> + if (nbytes >= sizeof(local_buffer)) {
> + buffer = kmalloc(nbytes + 1, GFP_KERNEL);
> + if (buffer == NULL)
> +   return -ENOMEM;
> + }
> + if (nbytes && copy_from_user(buffer, userbuf, nbytes))
> + return -EFAULT;
> +
> + buffer[nbytes] = 0; /* nul-terminate */
> + strstrip(buffer);
> + retval = cft->write_string(cgrp, cft, buffer);
> + if (!retval)
> + retval = nbytes;
> + if (buffer != local_buffer)
> + kfree(buffer);
> + return retval;

> +}
> +
> static ssize_t cgroup_common_file_write(struct cgroup *cgrp,
> struct cftype *cft,
> struct file *file,
> @@ -1440,6 +1473,8 @@ static ssize_t cgroup_file_write(struct
> return cft->write(cgrp, cft, file, buf, nbytes, ppos);
> if (cft->write_u64 || cft->write_s64)
> return cgroup_write_X64(cgrp, cft, file, buf, nbytes, ppos);
> + if (cft->write_string)
> + return cgroup_write_string(cgrp, cft, file, buf, nbytes, ppos);
> if (cft->trigger) {
> int ret = cft->trigger(cgrp, (unsigned int)cft->private);
> return ret ? ret : nbytes;
>
> --

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

---

Subject: Re: [PATCH 3/8] CGroup Files: Move the release_agent file to use typed handlers
Posted by akpm on Tue, 24 Jun 2008 23:23:25 GMT
View Forum Message <> Reply to Message

On Fri, 20 Jun 2008 16:44:01 -0700
menage@google.com wrote:

> Adds cgroup_release_agent_write() and cgroup_release_agent_show()
> methods to handle writing/reading the path to a cgroup hierarchy's
> release agent. As a result, cgroup_common_file_read() is now unnecessary.
>
> As part of the change, a previously-tolerated race in
> cgroup_release_agent() is avoided by copying the current
> release_agent_path prior to calling call_usermode_helper().
>
> Signed-off-by: Paul Menage <menage@google.com>
>
> ---
> include/linux/cgroup.h | 2
> kernel/cgroup.c | 125 ++++++++++++++++++++++++++++++++----------------
> 2 files changed, 59 insertions(+), 68 deletions(-)
>
> Index: cws-2.6.26-rc5-mm3/kernel/cgroup.c
> ==========
> --- cws-2.6.26-rc5-mm3.orig/kernel/cgroup.c

> +++ cws-2.6.26-rc5-mm3/kernel/cgroup.c
> @@ -89,11 +89,7 @@ struct cgroupfs_root {
> /* Hierarchy-specific flags */
> unsigned long flags;
>
> - /* The path to use for release notifications. No locking
> - * between setting and use - so if userspace updates this
> - * while child cgroups exist, you could miss a
> - * notification. We ensure that it's always a valid
> - * NUL-terminated string */
> + /* The path to use for release notifications. */
> char release_agent_path[PATH_MAX];
> };
>
> @@ -1329,6 +1325,45 @@ enum cgroup_filetype {
> FILE_RELEASE_AGENT,
> };
>
> +/**
> + * cgroup_lock_live_group - take cgroup_mutex and check that cgrp is alive.
> + * @cgrp: the cgroup to be checked for liveness
> + *
> + * Returns true (with lock held) on success, or false (with no lock
> + * held) on failure.
> + */
> +int cgroup_lock_live_group(struct cgroup *cgrp)
> +{
> + mutex_lock(&cgroup_mutex);
> + if (cgroup_is_removed(cgrp)) {
> + mutex_unlock(&cgroup_mutex);
> + return false;
> + }
> + return true;
> +}

I think that if we're going to do this it would be nice to add a
symmetrical cgroup_unlock_live_group()?

Because code like this:

> + if (!cgroup_lock_live_group(cgrp))
> + return -ENODEV;
> + strcpy(cgrp->root->release_agent_path, buffer);
> + mutex_unlock(&cgroup_mutex);

is a bit WTFish, no? it forces each caller of cgroup_lock_live_group()
to know about cgroup_lock_live_group() internals.

That would be kind of OKayish if this code was closely localised, but...

> --- cws-2.6.26-rc5-mm3.orig/include/linux/cgroup.h
> +++ cws-2.6.26-rc5-mm3/include/linux/cgroup.h
> @@ -295,6 +295,8 @@ int cgroup_add_files(struct cgroup *cgrp
>
> int cgroup_is_removed(const struct cgroup *cgrp);
>
> +int cgroup_lock_live_group(struct cgroup *cgrp);
> +
> int cgroup_path(const struct cgroup *cgrp, char *buf, int buflen);
>
> int cgroup_task_count(const struct cgroup *cgrp);
>

I assume this gets used in another .c file in a later patch.

Subject: Re: [PATCH 2/8] CGroup Files: Add write_string cgroup control file method
Posted by Paul Menage on Tue, 24 Jun 2008 23:26:21 GMT
View Forum Message <> Reply to Message

On Tue, Jun 24, 2008 at 4:19 PM, Andrew Morton
<akpm@linux-foundation.org> wrote:
>>   /*
>> +   * write_string() is passed a nul-terminated kernelspace
>> +   * buffer of maximum length determined by max_write_len.
>> +   * Returns 0 or -ve error code.
>> +   */
>> +   int (*write_string)(struct cgroup *cgrp, struct cftype *cft,
>> +   const char *buffer);
>
> Everything seems to use size_t (or ssize_t?) except for the ->write_string
> return value. Can any of this be improved?

What other things are you including as "everything"?

write_string() returns 0 on success or a -ve error code on failure -
it doesn't have the concept of writing some fraction of the passed
bytes.

The functions that deal in size_t/ssize_t (along with userspace
buffers, files and position pointers) are the glue that interfaces
with the filesystem layer. My aim (which is furthered by this patch

series) is to keep as much of that as possible in the cgroup layer
itself, and to reduce filesystem glue in the cgroup subsystems. The
raw file interface is still exposed by cgroups for those subsystems
that really need it, but it should be the exception rather than the
rule.

Paul

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

---

Subject: Re: [PATCH 3/8] CGroup Files: Move the release_agent file to use typed handlers
Posted by Paul Menage on Tue, 24 Jun 2008 23:30:35 GMT
View Forum Message <> Reply to Message

On Tue, Jun 24, 2008 at 4:23 PM, Andrew Morton
<akpm@linux-foundation.org> wrote:
>> +/**
>> + * cgroup_lock_live_group - take cgroup_mutex and check that cgrp is alive.
>> + * @cgrp: the cgroup to be checked for liveness
>> + *
>> + * Returns true (with lock held) on success, or false (with no lock
>> + * held) on failure.
>> + */
>> +int cgroup_lock_live_group(struct cgroup *cgrp)
>> +{
>> +   mutex_lock(&cgroup_mutex);
>> +   if (cgroup_is_removed(cgrp)) {
>> +   mutex_unlock(&cgroup_mutex);
>> +   return false;
>> +   }
>> +   return true;
>> +}
>
> I think that if we're going to do this it would be nice to add a
> symmetrical cgroup_unlock_live_group()?

There's already a cgroup_unlock() function exported in cgroup.h -
that's the counterpart to both cgroup_lock() and
cgroup_lock_live_group(). I can add a comment about this in the docs
for cgroup_lock_live_cgroup().

>
> Because code like this:

>
>> +	if (!cgroup_lock_live_group(cgrp))
>> +		return -ENODEV;
>> +	strcpy(cgrp->root->release_agent_path, buffer);
>> +	mutex_unlock(&cgroup_mutex);
>
> is a bit WTFish, no? it forces each caller of cgroup_lock_live_group()
> to know about cgroup_lock_live_group() internals.

cgroup_mutex isn't directly exported outside of cgroup.c, so real
callers would have no choice but to use cgroup_unlock() in this
instance. I guess it could make sense to be consistent and use
cgroup_unlock() within cgroup.c as well.

Paul