Subject: [RFC][PATCH 0/7] CGroup API: More structured API for CGroups control files
Posted by Paul Menage on Fri, 15 Feb 2008 20:44:18 GMT
View Forum Message <> Reply to Message

This set of patches makes the Control Groups API more structured and self-describing.

1) Allows control files to be associated with data types such as "u64", "string", "map", etc. These types show up in a new cgroup.api file in each cgroup directory, along with a user-readable string. Files that use cgroup-provided data accessors have these file types inferred automatically.

2) Moves various files in cpusets and the memory controller from using custom-written file handlers to cgroup-defined handlers

3) Adds the "cgroup." prefix for existing cgroup-provided control files (tasks, release_agent, releasable, notify_on_release). Given than we've already had 2.6.24 go out without this prefix, I guess this could be a little contentious - but it seems like a good move to prevent name clashes in the future. (Note that this doesn't affect mounting the legacy cpuset filesystem, since the compatibility layer disables all prefixes when mounted with filesystem type "cpuset"). If people object too strongly, we could just make this the case for *new* cgroup API files, but I think this is a case where consistency would be better than compatibility - I'd be surprised if anyone has written major legacy apps yet that rely on 2.6.24 cgroup control file names.

There are various motivations for this:

1) We said at Kernel Summit '07 that the cgroup API wouldn't be allowed to spiral into an arbitrary mess of ad-hoc APIs. Having simple ways to represent common data types makes this easier. (E.g. one standard way to report a map of string,u64 pairs to userspace.)

2) People were divided on the issue of binary APIs versus ASCII APIs for control groups. Compatibility with the existing cpusets system, and ease of experimentation, were two important reasons for going with the current. ASCII API. But by having structured control files, we can open the path towards having more efficient binary APIs for simpler and more efficient programmatic access too, without any additional modifications required from the subsystems themselves.

My plans for this potential binary API are a little hazy at this point, but they might go something like opening a cgroup.bin file in a cgroup directory, and writing the names of the control files that you

were interested in; then a read on that file handle would return the
contents of the given control files in a single read in a simple
binary format. (Better suggestions are welcome). Regardless, getting a
good typing/structure on the control files is an important first step
if we want to go in that direction.

3) The memory controller currently has files with the "_in_bytes"
suffix, on the grounds that otherwise it's not obvious to a new user
what they represent. By moving the description to a auto-generated API
file, we can remove this (IMO) inelegant suffix.

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

## Subject: [RFC][PATCH 1/7] CGroup API: Add cgroup.api control file
Posted by Paul Menage on Fri, 15 Feb 2008 20:44:19 GMT

View Forum Message <> Reply to Message

Add a cgroup.api control file in every cgroup directory. This reports
for each control file the type of data represented by that control
file, and a user-friendly description of the contents.

A secondary effect of this patch is to add the "cgroup." prefix in
front of all cgroup-provided control files. This will reduce the
chance of future control files clashing with user-provided names.

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

```
---
 include/linux/cgroup.h |   21 +++++++
 kernel/cgroup.c        |  133 ++++++++++++++++++++++++++++++++++++++++++++++---
 2 files changed, 148 insertions(+), 6 deletions(-)

Index: cgroupmap-2.6.24-mm1/include/linux/cgroup.h
===================================================================
--- cgroupmap-2.6.24-mm1.orig/include/linux/cgroup.h
+++ cgroupmap-2.6.24-mm1/include/linux/cgroup.h
@@ -179,12 +179,33 @@ struct css_set {
  * - the 'cftype' of the file is file->f_dentry->d_fsdata
  */

+/*
+ * The various types of control file that are reported in the
+ * cgroup.api file. "String" is a catch-all default, but should only
+ * be used for special cases. If you use the appropriate accessors
+ * (such as "read_uint") in your control file, then you can leave this
+ * as 0 (CGROUP_FILE_UNKNOWN) and let cgroup figure out the right type.
+ */
+enum cgroup_file_type {
+ CGROUP_FILE_UNKNOWN = 0,
+ CGROUP_FILE_VOID,
+ CGROUP_FILE_U64,
+ CGROUP_FILE_STRING,
+};
+
 #define MAX_CFTYPE_NAME 64
 struct cftype {
  /* By convention, the name should begin with the name of the
   * subsystem, followed by a period */
  char name[MAX_CFTYPE_NAME];
  int private;
+
+ /* The type of a file - reported in the cgroup.api file */
+ enum cgroup_file_type type;
+
+ /* Human-readable description of the file */
+ const char *desc;
+
  int (*open) (struct inode *inode, struct file *file);
  ssize_t (*read) (struct cgroup *cont, struct cftype *cft,
     struct file *file,
Index: cgroupmap-2.6.24-mm1/kernel/cgroup.c
===================================================================
--- cgroupmap-2.6.24-mm1.orig/kernel/cgroup.c
+++ cgroupmap-2.6.24-mm1/kernel/cgroup.c
@@ -1301,6 +1301,7 @@ enum cgroup_filetype {
  FILE_NOTIFY_ON_RELEASE,
  FILE_RELEASABLE,
  FILE_RELEASE_AGENT,
+ FILE_API,
 };

 static ssize_t cgroup_write_uint(struct cgroup *cgrp, struct cftype *cft,
@@ -1611,17 +1612,21 @@ static int cgroup_create_dir(struct cgro
 }

 int cgroup_add_file(struct cgroup *cgrp,
-         struct cgroup_subsys *subsys,
-         const struct cftype *cft)
+      struct cgroup_subsys *subsys,
+      const struct cftype *cft)
```

```
 {
  struct dentry *dir = cgrp->dentry;
  struct dentry *dentry;
  int error;

  char name[MAX_CGROUP_TYPE_NAMELEN + MAX_CFTYPE_NAME + 2] = { 0 };
- if (subsys && !test_bit(ROOT_NOPREFIX, &cgrp->root->flags)) {
-  strcpy(name, subsys->name);
-  strcat(name, ".");
+ if (!test_bit(ROOT_NOPREFIX, &cgrp->root->flags)) {
+  if (subsys) {
+   strcpy(name, subsys->name);
+   strcat(name, ".");
+  } else {
+   strcpy(name, "cgroup.");
+  }
  }
  strcat(name, cft->name);
  BUG_ON(!mutex_is_locked(&dir->d_inode->i_mutex));
@@ -2126,6 +2131,110 @@ static u64 cgroup_read_releasable(struct
  return test_bit(CGRP_RELEASABLE, &cgrp->flags);
 }

+static const struct file_operations cgroup_api_file_operations = {
+ .read = seq_read,
+ .llseek = seq_lseek,
+ .release = seq_release,
+};
+
+/*
+ * cgroup.api is a file in each cgroup directory that gives the types
+ * and descriptions of the various control files in that directory.
+ */
+
+static struct dentry *cgroup_api_advance(struct dentry *d, int advance)
+{
+ struct dentry *parent = d->d_parent;
+ struct list_head *l = &d->d_u.d_child;
+ while (true) {
+  if (advance)
+   l = l->next;
+  advance = true;
+  /* Did we reach the end of the directory? */
+  if (l == &parent->d_subdirs)
+   return NULL;
+  d = container_of(l, struct dentry, d_u.d_child);
+  /* Skip cgroup subdirectories */
+  if (d->d_inode && S_ISREG(d->d_inode->i_mode))
+   return d;
+ }
+}
+
+static void *cgroup_api_start(struct seq_file *sf, loff_t *pos)
+{
+ struct dentry *parent = sf->private;
+ struct dentry *d;
+ loff_t l = 0;
+ spin_lock(&dcache_lock);
+ if (list_empty(&parent->d_subdirs))
+  return NULL;
+ d = container_of(parent->d_subdirs.next, struct dentry, d_u.d_child);
+ d = cgroup_api_advance(d, 0);
+ while (d && l < *pos) {
+  (*pos)++;
+  d = cgroup_api_advance(d, 1);
+ }
+
+ return d;
+}
+
+static void *cgroup_api_next(struct seq_file *sf, void *v, loff_t *pos)
+{
+ struct dentry *d = v;
+ (*pos)++;
+ return cgroup_api_advance(d, 1);
+}
+
+static void cgroup_api_stop(struct seq_file *sf, void *v)
+{
+ spin_unlock(&dcache_lock);
+}
+
+static const char *cft_type_names[] = {
+ "unknown",
+ "void",
+ "u64",
+ "string",
+ "u64map",
+};
+
+static int cgroup_api_show(struct seq_file *sf, void *v)
+{
+ struct dentry *d = v;
+ struct cftype *cft = __d_cft(d);
+ unsigned type = cft->type;
+ if (type == CGROUP_FILE_UNKNOWN) {
```

```
+  if (cft->read_uint)
+   type = CGROUP_FILE_U64;
+  else if (cft->read)
+   type = CGROUP_FILE_STRING;
+  else if (!cft->open)
+   type = CGROUP_FILE_VOID;
+ }
+ if (type >= ARRAY_SIZE(cft_type_names))
+  type = CGROUP_FILE_UNKNOWN;
+ return seq_printf(sf, "%s\t%s\t%s\n", d->d_name.name,
+     cft_type_names[type], cft->desc ?: "");
+}
+
+static const struct seq_operations cgroup_api_seqop = {
+ .start = cgroup_api_start,
+ .next = cgroup_api_next,
+ .stop = cgroup_api_stop,
+ .show = cgroup_api_show,
+};
+
+static int cgroup_api_open(struct inode *inode, struct file *file)
+{
+ int ret = seq_open(file, &cgroup_api_seqop);
+ if (ret == 0) {
+  struct seq_file *sf = file->private_data;
+  sf->private = file->f_dentry->d_parent;
+  file->f_op = &cgroup_api_file_operations;
+ }
+ return ret;
+}
+
 /*
  * for the common functions, 'private' gives the type of file
  */
@@ -2137,6 +2246,7 @@ static struct cftype files[] = {
   .write = cgroup_common_file_write,
   .release = cgroup_tasks_release,
   .private = FILE_TASKLIST,
+  .desc = "Thread ids of threads in this cgroup",
  },

  {
@@ -2144,13 +2254,23 @@ static struct cftype files[] = {
   .read_uint = cgroup_read_notify_on_release,
   .write = cgroup_common_file_write,
   .private = FILE_NOTIFY_ON_RELEASE,
+  .desc =
+  "Should the release agent trigger when this cgroup is empty",
  },

  {
   .name = "releasable",
   .read_uint = cgroup_read_releasable,
   .private = FILE_RELEASABLE,
- }
+  .desc = "Is this cgroup able to be freed when empty"
+ },
+
+ {
+  .name = "api",
+  .open = cgroup_api_open,
+  .private = FILE_API,
+  .desc = "Control file descriptions",
+ },
 };

 static struct cftype cft_release_agent = {
@@ -2158,6 +2278,7 @@ static struct cftype cft_release_agent =
  .read = cgroup_common_file_read,
  .write = cgroup_common_file_write,
  .private = FILE_RELEASE_AGENT,
+ .desc = "Path to release agent binary",
 };

 static int cgroup_populate_dir(struct cgroup *cgrp)

--
```

### Subject: [RFC][PATCH 2/7] CGroup API: Add cgroup map data type
### Posted by Paul Menage on Fri, 15 Feb 2008 20:44:20 GMT

View Forum Message <> Reply to Message

Adds a new type of supported control file representation, a map from
strings to u64 values.

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

```
---
 include/linux/cgroup.h |   19 +++++++++++++++
 kernel/cgroup.c        |   61 ++++++++++++++++++++++++++++++++++++++++++++++++++-
 2 files changed, 79 insertions(+), 1 deletion(-)
```

```
Index: cgroupmap-2.6.24-mm1/include/linux/cgroup.h
===================================================================
--- cgroupmap-2.6.24-mm1.orig/include/linux/cgroup.h
+++ cgroupmap-2.6.24-mm1/include/linux/cgroup.h
@@ -191,6 +191,17 @@ enum cgroup_file_type {
  CGROUP_FILE_VOID,
  CGROUP_FILE_U64,
  CGROUP_FILE_STRING,
+ CGROUP_FILE_MAP,
+};
+
+/*
+ * cgroup_map_cb is an abstract callback API for reporting map-valued
+ * control files
+ */
+
+struct cgroup_map_cb {
+ int (*fill)(struct cgroup_map_cb *cb, const char *key, u64 value);
+ void *state;
 };

 #define MAX_CFTYPE_NAME 64
@@ -215,6 +226,14 @@ struct cftype {
   * single integer. Use it in place of read()
   */
  u64 (*read_uint) (struct cgroup *cont, struct cftype *cft);
+ /*
+  * read_map() is used for defining a map of key/value
+  * pairs. It should call cb->fill(cb, key, value) for each
+  * entry.
+  */
+ int (*read_map) (struct cgroup *cont, struct cftype *cft,
+    struct cgroup_map_cb *cb);
+
  ssize_t (*write) (struct cgroup *cont, struct cftype *cft,
      struct file *file,
      const char __user *buf, size_t nbytes, loff_t *ppos);
Index: cgroupmap-2.6.24-mm1/kernel/cgroup.c
===================================================================
--- cgroupmap-2.6.24-mm1.orig/kernel/cgroup.c
+++ cgroupmap-2.6.24-mm1/kernel/cgroup.c
@@ -1488,6 +1488,46 @@ static ssize_t cgroup_file_read(struct f
  return -EINVAL;
 }

+/*
+ * seqfile ops/methods for returning structured data. Currently just
+ * supports string->u64 maps, but can be extended in future.
+ */
+
+struct cgroup_seqfile_state {
+ struct cftype *cft;
+ struct cgroup *cgroup;
+};
+
+static int cgroup_map_add(struct cgroup_map_cb *cb, const char *key, u64 value)
+{
+ struct seq_file *sf = cb->state;
+ return seq_printf(sf, "%s: %llu\n", key, value);
+}
+
+static int cgroup_seqfile_show(struct seq_file *m, void *arg)
+{
+ struct cgroup_seqfile_state *state = m->private;
+ struct cftype *cft = state->cft;
+ struct cgroup_map_cb cb = {
+  .fill = cgroup_map_add,
+  .state = m,
+ };
+ if (cft->read_map) {
+  return cft->read_map(state->cgroup, cft, &cb);
+ } else {
+  BUG();
+ }
+}
+
+int cgroup_seqfile_release(struct inode *inode, struct file *file)
+{
+ struct seq_file *seq = file->private_data;
+ kfree(seq->private);
+ return single_release(inode, file);
+}
+
+static struct file_operations cgroup_seqfile_operations;
+
 static int cgroup_file_open(struct inode *inode, struct file *file)
 {
  int err;
@@ -1500,7 +1540,18 @@ static int cgroup_file_open(struct inode
  cft = __d_cft(file->f_dentry);
  if (!cft)
   return -ENODEV;
- if (cft->open)
+ if (cft->read_map) {
+  struct cgroup_seqfile_state *state =
+   kzalloc(sizeof(*state), GFP_USER);
+  if (!state)
+   return -ENOMEM;
+  state->cft = cft;
+  state->cgroup = __d_cgrp(file->f_dentry->d_parent);
+  file->f_op = &cgroup_seqfile_operations;
+  err = single_open(file, cgroup_seqfile_show, state);
+  if (err < 0)
+   kfree(state);
+ } else if (cft->open)
   err = cft->open(inode, file);
  else
   err = 0;
@@ -1539,6 +1590,12 @@ static struct file_operations cgroup_fil
  .release = cgroup_file_release,
 };

+static struct file_operations cgroup_seqfile_operations = {
+ .read = seq_read,
+ .llseek = seq_lseek,
+ .release = cgroup_seqfile_release,
+};
+
 static struct inode_operations cgroup_dir_inode_operations = {
  .lookup = simple_lookup,
  .mkdir = cgroup_mkdir,
@@ -2206,6 +2263,8 @@ static int cgroup_api_show(struct seq_fi
  if (type == CGROUP_FILE_UNKNOWN) {
   if (cft->read_uint)
    type = CGROUP_FILE_U64;
+  else if (cft->read_map)
+   type = CGROUP_FILE_MAP;
   else if (cft->read)
    type = CGROUP_FILE_STRING;
   else if (!cft->open)
```

--

_______________________________________________

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

### Subject: [RFC][PATCH 5/7] CGroup API: Use read_uint in memory controller

Posted by Paul Menage on Fri, 15 Feb 2008 20:44:23 GMT

View Forum Message <> Reply to Message

Update the memory controller to use read_uint for its

limit/usage/failcnt control files, calling the new
res_counter_read_uint() function. This allows the files to show up as
u64 rather than string in the cgroup.api file.

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

```
---
 mm/memcontrol.c |   15 ++++++---------
 1 file changed, 6 insertions(+), 9 deletions(-)

Index: cgroupmap-2.6.24-mm1/mm/memcontrol.c
===================================================================
--- cgroupmap-2.6.24-mm1.orig/mm/memcontrol.c
+++ cgroupmap-2.6.24-mm1/mm/memcontrol.c
@@ -922,13 +922,10 @@ int mem_cgroup_write_strategy(char *buf,
  return 0;
 }

-static ssize_t mem_cgroup_read(struct cgroup *cont,
-   struct cftype *cft, struct file *file,
-   char __user *userbuf, size_t nbytes, loff_t *ppos)
+static u64 mem_cgroup_read(struct cgroup *cont, struct cftype *cft)
 {
- return res_counter_read(&mem_cgroup_from_cont(cont)->res,
-    cft->private, userbuf, nbytes, ppos,
-    NULL);
+ return res_counter_read_uint(&mem_cgroup_from_cont(cont)->res,
+         cft->private);
 }

 static ssize_t mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
@@ -1006,18 +1003,18 @@ static struct cftype mem_cgroup_files[]
  {
   .name = "usage_in_bytes",
   .private = RES_USAGE,
-  .read = mem_cgroup_read,
+  .read_uint = mem_cgroup_read,
  },
  {
   .name = "limit_in_bytes",
   .private = RES_LIMIT,
   .write = mem_cgroup_write,
-  .read = mem_cgroup_read,
+  .read_uint = mem_cgroup_read,
  },
  {
   .name = "failcnt",
   .private = RES_FAILCNT,
-  .read = mem_cgroup_read,
+  .read_uint = mem_cgroup_read,
  },
  {
   .name = "force_empty",
```

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

## Subject: [RFC][PATCH 6/7] CGroup API: Use descriptions for memory controller API files

Posted by Paul Menage on Fri, 15 Feb 2008 20:44:24 GMT

View Forum Message <> Reply to Message

This patch adds descriptions to the memory controller API files to
indicate that the usage/limit are in bytes; the names of the control
files can then be simplified to usage/limit.

Also removes the unnecessary mem_force_empty_read() function

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

```
---
 mm/memcontrol.c |   21 +++++----------------
 1 file changed, 5 insertions(+), 16 deletions(-)

Index: cgroupmap-2.6.24-mm1/mm/memcontrol.c
===================================================================
--- cgroupmap-2.6.24-mm1.orig/mm/memcontrol.c
+++ cgroupmap-2.6.24-mm1/mm/memcontrol.c
@@ -950,19 +950,6 @@ static ssize_t mem_force_empty_write(str
  return ret;
 }

-/*
- * Note: This should be removed if cgroup supports write-only file.
- */
-
-static ssize_t mem_force_empty_read(struct cgroup *cont,
-    struct cftype *cft,
-    struct file *file, char __user *userbuf,
-    size_t nbytes, loff_t *ppos)
-{
- return -EINVAL;
```

```
-}
-
-
 static const struct mem_cgroup_stat_desc {
  const char *msg;
  u64 unit;
@@ -1001,15 +988,17 @@ static int mem_control_stat_show(struct

 static struct cftype mem_cgroup_files[] = {
  {
-  .name = "usage_in_bytes",
+  .name = "usage",
   .private = RES_USAGE,
   .read_uint = mem_cgroup_read,
+  .desc = "Memory usage in bytes",
  },
  {
-  .name = "limit_in_bytes",
+  .name = "limit",
   .private = RES_LIMIT,
   .write = mem_cgroup_write,
   .read_uint = mem_cgroup_read,
+  .desc = "Memory limit in bytes",
  },
  {
   .name = "failcnt",
@@ -1019,7 +1008,7 @@ static struct cftype mem_cgroup_files[]
  {
   .name = "force_empty",
   .write = mem_force_empty_write,
-  .read = mem_force_empty_read,
+  .desc = "Write to this file to forget all memory charges"
  },
  {
   .name = "stat",

--
```

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

## Subject: Re: [RFC][PATCH 0/7] CGroup API: More structured API for CGroups control files
Posted by KAMEZAWA Hiroyuki on Sat, 16 Feb 2008 04:20:05 GMT
View Forum Message <> Reply to Message

On Fri, 15 Feb 2008 12:44:18 -0800
Paul Menage <menage@google.com> wrote:

> 
> This set of patches makes the Control Groups API more structured and
> self-describing.
> 
> 1) Allows control files to be associated with data types such as
> "u64", "string", "map", etc. These types show up in a new cgroup.api
> file in each cgroup directory, along with a user-readable
> string. Files that use cgroup-provided data accessors have these file
> types inferred automatically.
> 
> 2) Moves various files in cpusets and the memory controller from using
> custom-written file handlers to cgroup-defined handlers
> 
> 3) Adds the "cgroup." prefix for existing cgroup-provided control
> files (tasks, release_agent, releasable, notify_on_release). Given
> than we've already had 2.6.24 go out without this prefix, I guess this
> could be a little contentious - but it seems like a good move to
> prevent name clashes in the future. (Note that this doesn't affect
> mounting the legacy cpuset filesystem, since the compatibility layer
> disables all prefixes when mounted with filesystem type "cpuset"). If
> people object too strongly, we could just make this the case for *new*
> cgroup API files, but I think this is a case where consistency would
> be better than compatibility - I'd be surprised if anyone has written
> major legacy apps yet that rely on 2.6.24 cgroup control file names.
>

Hi,  I like this direction very much. thank you for your work.
Self-describing cgroup.api file is a good idea!

One request from me is add "mode" bit to cftype for allowing
write-only/read-only files.

Thanks,
-Kame

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

Subject: Re: [RFC][PATCH 0/7] CGroup API: More structured API for CGroups control files
Posted by Li Zefan on Sat, 16 Feb 2008 09:31:50 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> On Fri, 15 Feb 2008 12:44:18 -0800
> Paul Menage <menage@google.com> wrote:
>
>> This set of patches makes the Control Groups API more structured and
>> self-describing.
>>
>> 1) Allows control files to be associated with data types such as
>> "u64", "string", "map", etc. These types show up in a new cgroup.api
>> file in each cgroup directory, along with a user-readable
>> string. Files that use cgroup-provided data accessors have these file
>> types inferred automatically.
>>
>> 2) Moves various files in cpusets and the memory controller from using
>> custom-written file handlers to cgroup-defined handlers
>>
>> 3) Adds the "cgroup." prefix for existing cgroup-provided control
>> files (tasks, release_agent, releasable, notify_on_release). Given
>> than we've already had 2.6.24 go out without this prefix, I guess this
>> could be a little contentious - but it seems like a good move to
>> prevent name clashes in the future. (Note that this doesn't affect
>> mounting the legacy cpuset filesystem, since the compatibility layer
>> disables all prefixes when mounted with filesystem type "cpuset"). If
>> people object too strongly, we could just make this the case for *new*
>> cgroup API files, but I think this is a case where consistency would
>> be better than compatibility - I'd be surprised if anyone has written
>> major legacy apps yet that rely on 2.6.24 cgroup control file names.
>>
>
>
> Hi,  I like this direction very much. thank you for your work.
> Self-describing cgroup.api file is a good idea!
>
> One request from me is add "mode" bit to cftype for allowing
> write-only/read-only files.
>
> Thanks,
> -Kame
>

I don't quite catch what you mean. Cgoup does support write-only/read-only
files. For a write-only file, just set .write and .write_uint to be NULL,
similar for a read-only file.

Do I miss something?

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

## Subject: Re: [RFC][PATCH 1/7] CGroup API: Add cgroup.api control file
## Posted by Balbir Singh on Sat, 16 Feb 2008 10:07:06 GMT

View Forum Message <> Reply to Message

Paul Menage wrote:

Hi, Paul,

Do we need to use a cgroup.api file? Why not keep up to date documentation and
get users to use that. I fear that, cgroup.api will not be kept up-to-date,
leading to confusion.

Why should the kernel carry so much of documentation in the image as strings?

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

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

## Subject: Re: [RFC][PATCH 0/7] CGroup API: More structured API for CGroups control files
## Posted by Paul Menage on Sat, 16 Feb 2008 17:40:18 GMT

View Forum Message <> Reply to Message

On Feb 16, 2008 1:31 AM, Li Zefan <lizf@cn.fujitsu.com> wrote:
>
> I don't quite catch what you mean. Cgoup does support write-only/read-only
> files. For a write-only file, just set .write and .write_uint to be NULL,
> similar for a read-only file.
>
> Do I miss something?
>

I suppose we could infer from the lack of any write handlers that we should give the file in the filesystem a mode of 444 rather 644.

Paul

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

## Subject: Re: [RFC][PATCH 1/7] CGroup API: Add cgroup.api control file
## Posted by Paul Menage on Sat, 16 Feb 2008 17:44:40 GMT

View Forum Message <> Reply to Message

On Feb 16, 2008 2:07 AM, Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
> Paul Menage wrote:
>
> Hi, Paul,
>
> Do we need to use a cgroup.api file? Why not keep up to date documentation and
> get users to use that. I fear that, cgroup.api will not be kept up-to-date,
> leading to confusion.

The cgroup.api file isn't meant to give complete documentation for a control file, simply a brief indication of its usage.

The aim is that most bits of the information reported in cgroup.api are auto-generated, so there shouldn't be problems with it getting out-of-date.

Is it just the space used by the documentation string that you're objecting to? The other function of the file is to declare a type for each variable.

Paul

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

## Subject: Re: [RFC][PATCH 1/7] CGroup API: Add cgroup.api control file
## Posted by Li Zefan on Mon, 18 Feb 2008 09:45:12 GMT

View Forum Message <> Reply to Message

Paul Menage wrote:
> On Feb 16, 2008 2:07 AM, Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
>> Paul Menage wrote:

>>
>> Hi, Paul,
>>
>> Do we need to use a cgroup.api file? Why not keep up to date documentation and
>> get users to use that. I fear that, cgroup.api will not be kept up-to-date,
>> leading to confusion.
>
> The cgroup.api file isn't meant to give complete documentation for a
> control file, simply a brief indication of its usage.
>

But we don't have /proc/proc.api or /sys/sysfs.api ...

> The aim is that most bits of the information reported in cgroup.api
> are auto-generated, so there shouldn't be problems with it getting
> out-of-date.
>
> Is it just the space used by the documentation string that you're
> objecting to? The other function of the file is to declare a type for
> each variable.
>

```
[root@localhost mnt]# cat cgroup.api
debug.current_css_set_refcount  u64
debug.current_css_set   u64
debug.taskcount u64
debug.cgroup_refcount   u64
cgroup.release_agent    string  Path to release agent binary
cgroup.api      unknown Control file descriptions
cgroup.releasable       u64     Is this cgroup able to be freed when empty
cgroup.notify_on_release        u64     Should the release agent trigger when this cgroup is empty
cgroup.tasks    string  Thread ids of threads in this cgroup
```

It seems to me this is a little messy.

And is it better to describe the debug subsystem too?

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

Subject: Re: [RFC][PATCH 1/7] CGroup API: Add cgroup.api control file
Posted by Balbir Singh on Mon, 18 Feb 2008 10:32:59 GMT
View Forum Message <> Reply to Message

Li Zefan wrote:

> Paul Menage wrote:
>> On Feb 16, 2008 2:07 AM, Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
>>> Paul Menage wrote:
>>>
>>> Hi, Paul,
>>>
>>> Do we need to use a cgroup.api file? Why not keep up to date documentation and
>>> get users to use that. I fear that, cgroup.api will not be kept up-to-date,
>>> leading to confusion.
>> The cgroup.api file isn't meant to give complete documentation for a
>> control file, simply a brief indication of its usage.
>>
>
> But we don't have /proc/proc.api or /sys/sysfs.api ...
>

Yes, doing that would make the size of the kernel go out of control.

>> The aim is that most bits of the information reported in cgroup.api
>> are auto-generated, so there shouldn't be problems with it getting
>> out-of-date.
>>
>> Is it just the space used by the documentation string that you're
>> objecting to? The other function of the file is to declare a type for
>> each variable.
>>

I like the documentation aspect, it is the space that I object to.

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

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

## Subject: Re: [RFC][PATCH 1/7] CGroup API: Add cgroup.api control file
## Posted by Paul Jackson on Tue, 19 Feb 2008 21:57:42 GMT

View Forum Message <> Reply to Message

Li Zefan wrote:
> It seems to me this is a little messy.

Agreed.  It looks too finicky to base real software on;  that is, I
doubt that any robust application or user level software is going to
depend on it for serious automated self-configuration.

I haven't seen a serious problem with cpuset documentation, which is
the earlier API of this flavor (though, granted, I'd be the last
person on the planet to see such ;).  It seems to me that this style
of API is easy enough for a variety of people to figure out that this
won't help that much.  And certainly the more interesting details that
need documentation are far too verbose for this presentation.

So ... little or no programmatic use, little or no human use.

It also reminds me a bit, in a very loose way, of efforts in sysfs that
have taken us a while, and too many changes, to get right.  One needs to
be -selective- in what one exposes, in order to minimize maintenance costs
and maximize the signal-to-noise ratio, over time.  This feels like it
exposes too much.

Finally, it goes against the one thingie per file (at most, one scalar
vector) that has worked well for us when tried.

As to the motivations Paul M gives:
 1) Avoid "an arbitrary mess of ad-hoc APIs":
 We can still do that, whether or not we "self-document" these
 API's in this manner.
 2) binary APIs versus ASCII APIs:
 Well, I have an ASCII API bias, not surprising.  But I'd
 suggest not doing things "in anticipation" of some future
 fuzzy binary API support.  Wait until that day actually arrives.
 3) The memory controller currently has files with the "_in_bytes":
 The traditional way to handle this is Documentation and man
 pages; good enough for my granddad, good enough for me ;).

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

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

## Subject: Re: [RFC][PATCH 1/7] CGroup API: Add cgroup.api control file
Posted by Paul Menage on Wed, 20 Feb 2008 02:51:29 GMT

View Forum Message <> Reply to Message

On Feb 19, 2008 1:57 PM, Paul Jackson <pj@sgi.com> wrote:
>
> Finally, it goes against the one thingie per file (at most, one scalar
> vector) that has worked well for us when tried.

Right, I like the idea of keeping things simple. But if you're going
to accept that a vector is useful, then it seems reasonable that some
other *simple* structured datatypes can be useful. An N-element
key/value map (a la /proc/meminfo) is, I think, nicer than having to
read values from N separate files.

>
> As to the motivations Paul M gives:
>  1) Avoid "an arbitrary mess of ad-hoc APIs":
>        We can still do that, whether or not we "self-document" these
>        API's in this manner.

We can, but this file makes it more clear what control files have a
well-defined API and which are just returning some ad-hoc string.

I guess it's not essential, I just figured that if we had that
information, it made sense to make it available to userspace. I guess
I'm happy with dropping the actual exposed cgroup.api file for now as
long as we can work towards reducing the number of control files that
just return strings, and make use of the structured output such as
read_uint() miore.

>  2) binary APIs versus ASCII APIs:
>        Well, I have an ASCII API bias, not surprising.  But I'd
>        suggest not doing things "in anticipation" of some future
>        fuzzy binary API support.  Wait until that day actually arrives.

I have a reasonably clear idea of how we can do the binary API.

That's mostly for a separate RFC. But for example, reading a map via
the binary API would be able to just return a list values since the
keys could be parsed once from the ascii map (provided that the
subsystem guaranteed that the map keys and their order wouldn't change
between reboots).

>  3) The memory controller currently has files with the "_in_bytes":
>        The traditional way to handle this is Documentation and man
>        pages; good enough for my granddad, good enough for me ;).

I've tried submitting patches to remove the in_bytes suffix and just
rely on the documentation, and people didn't seem to like it ...

Paul

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

## Subject: Re: [RFC][PATCH 1/7] CGroup API: Add cgroup.api control file
## Posted by Paul Menage on Wed, 20 Feb 2008 02:51:43 GMT

View Forum Message <> Reply to Message

On Feb 18, 2008 1:45 AM, Li Zefan <lizf@cn.fujitsu.com> wrote:
> >
>
> But we don't have /proc/proc.api or /sys/sysfs.api ...

True. And /proc is a bit of a mess. Having a similar API file for
sysfs sounds like a good idea to me.

>
> And is it better to describe the debug subsystem too?
>

Yes, probably, but that would be a separate patch to the debug
subsystem itself, not the main cgroups code.

Paul

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

## Subject: Re: [RFC][PATCH 1/7] CGroup API: Add cgroup.api control file
## Posted by Paul Jackson on Wed, 20 Feb 2008 05:17:26 GMT

View Forum Message <> Reply to Message

Paul M wrote:
> I guess it's not essential, I just figured that if we had that
> information, it made sense to make it available to userspace. I guess
> I'm happy with dropping the actual exposed cgroup.api file for now as
> long as we can work towards reducing the number of control files that
> just return strings, and make use of the structured output such as
> read_uint() miore.

I could certainly go along with that ... reducing the proportion of
control files returning untyped strings.

My sense of kernel-user API's is that usually the less said the better. Identify the essential information that one side requires from the other via a runtime API, and pass only that.  API's represent a lifetime commitment, so the less promised the better.

Perhaps my primary concern with these *.api files was that I did not understand who or what the critical use or user was; who found this essential, not just nice to have.

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

---

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

---

## Subject: Re: [RFC][PATCH 1/7] CGroup API: Add cgroup.api control file
## Posted by Paul Menage on Wed, 20 Feb 2008 05:23:33 GMT

View Forum Message <> Reply to Message

On Feb 19, 2008 9:17 PM, Paul Jackson <pj@sgi.com> wrote:
>
> Perhaps my primary concern with these *.api files was that I did not
> understand who or what the critical use or user was; who found this
> essential, not just nice to have.
>

Right now, no-one would find it essential. If/when a binary API is added, I guess I'll ressurrect this part of the patchset.

Paul

---

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