
Subject: [PATCH v8 4/3] cgroups: use flex_array in attach_proc
Posted by [Ben Blum](#) on Wed, 16 Feb 2011 19:22:00 GMT
[View Forum Message](#) <> [Reply to Message](#)

Convert cgroup_attach_proc to use flex_array.

From: Ben Blum <bblum@andrew.cmu.edu>

The cgroup_attach_proc implementation requires a pre-allocated array to store task pointers to atomically move a thread-group, but asking for a monolithic array with kmalloc() may be unreliable for very large groups. Using flex_array provides the same functionality with less risk of failure.

This is a post-patch for cgroup-procs-write.patch.

Signed-off-by: Ben Blum <bblum@andrew.cmu.edu>

```
---
kernel/cgroup.c | 37 ++++++-----
1 files changed, 28 insertions(+), 9 deletions(-)

diff --git a/kernel/cgroup.c b/kernel/cgroup.c
index 58b364a..feba784 100644
--- a/kernel/cgroup.c
+++ b/kernel/cgroup.c
@@ -57,6 +57,7 @@
#include <linux/vmalloc.h> /* TODO: replace with more sophisticated array */
#include <linux/eventfd.h>
#include <linux/poll.h>
+#include <linux/flex_array.h> /* used in cgroup_attach_proc */

#include <asm/atomic.h>

@@ -1985,7 +1986,7 @@ int cgroup_attach_proc(struct cgroup *cgrp, struct task_struct *leader)
    struct cgroupfs_root *root = cgrp->root;
    /* threadgroup list cursor and array */
    struct task_struct *tsk;
- struct task_struct **group;
+ struct flex_array *group;
    /*
     * we need to make sure we have css_sets for all the tasks we're
     * going to move -before- we actually start moving them, so that in
@@ -2002,9 +2003,15 @@ int cgroup_attach_proc(struct cgroup *cgrp, struct task_struct
*leader)
    * and if threads exit, this will just be an over-estimate.
    */
    group_size = get_nr_threads(leader);
- group = kmalloc(group_size * sizeof(*group), GFP_KERNEL);
+ /* flex_array supports very large thread-groups better than kmalloc. */
```

```

+ group = flex_array_alloc(sizeof(struct task_struct *), group_size,
+   GFP_KERNEL);
+   if (!group)
+       return -ENOMEM;
+ /* pre-allocate to guarantee space while iterating in rcu read-side. */
+ retval = flex_array_prealloc(group, 0, group_size - 1, GFP_KERNEL);
+ if (retval)
+   goto out_free_group_list;

/* prevent changes to the threadgroup list while we take a snapshot. */
rcu_read_lock();
@@ -2027,7 +2034,12 @@ int cgroup_attach_proc(struct cgroup *cgrp, struct task_struct
*leader)
/* as per above, nr_threads may decrease, but not increase. */
BUG_ON(i >= group_size);
get_task_struct(tsk);
- group[i] = tsk;
+ /*
+  * saying GFP_ATOMIC has no effect here because we did prealloc
+  * earlier, but it's good form to communicate our expectations.
+  */
+ retval = flex_array_put_ptr(group, i, tsk, GFP_ATOMIC);
+ BUG_ON(retval != 0);
+   i++;
+ } while_each_thread(leader, tsk);
/* remember the number of threads in the array for later. */
@@ -2050,7 +2062,9 @@ int cgroup_attach_proc(struct cgroup *cgrp, struct task_struct *leader)
if (ss->can_attach_task) {
/* run on each task in the threadgroup. */
for (i = 0; i < group_size; i++) {
-   retval = ss->can_attach_task(cgrp, group[i]);
+   tsk = flex_array_get_ptr(group, i);
+   BUG_ON(tsk == NULL);
+   retval = ss->can_attach_task(cgrp, tsk);
+   if (retval) {
+       failed_ss = ss;
+       goto out_cancel_attach;
+   }
}
@@ -2065,7 +2079,8 @@ int cgroup_attach_proc(struct cgroup *cgrp, struct task_struct *leader)
*/
INIT_LIST_HEAD(&newcgrp_list);
for (i = 0; i < group_size; i++) {
-   tsk = group[i];
+   tsk = flex_array_get_ptr(group, i);
+   BUG_ON(tsk == NULL);
+   /* nothing to do if this task is already in the cgroup */
+   oldcgrp = task_cgroup_from_root(tsk, root);
+   if (cgrp == oldcgrp)
}
@@ -2104,7 +2119,8 @@ int cgroup_attach_proc(struct cgroup *cgrp, struct task_struct *leader)

```

```

    ss->pre_attach(cgrp);
}
for (i = 0; i < group_size; i++) {
- tsk = group[i];
+ tsk = flex_array_get_ptr(group, i);
+ BUG_ON(tsk == NULL);
    /* leave current thread as it is if it's already there */
    oldcgrp = task_cgroup_from_root(tsk, root);
    if (cgrp == oldcgrp)
@@ -2154,10 +2170,13 @@ out_cancel_attach:
}
}
/* clean up the array of referenced threads in the group. */
- for (i = 0; i < group_size; i++)
- put_task_struct(group[i]);
+ for (i = 0; i < group_size; i++) {
+ tsk = flex_array_get_ptr(group, i);
+ BUG_ON(tsk == NULL);
+ put_task_struct(tsk);
+ }
out_free_group_list:
- kfree(group);
+ flex_array_free(group);
return retval;
}

```

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>
