Subject: [PATCH 11/11] restart: account for all ghost sids before ghost pgids
Posted by Oren Laadan on Mon, 07 Feb 2011 17:21:32 GMT
View Forum Message <> Reply to Message

This patch fixes the following two bugs:

1) Consider a subtree checkpoint of one task with pid != sid and also pgid == sid. At restart, ckpt_init_tree() will add a ghost task that accounts for the pgid, and ckpt_setup_task() will set its ppid and sid to be the root task. But because the root task is _not_ a session leader, ckpt_set_creator() will incorrecly add a dead task between the root and the ghost.

We fix this by ensuring that in this case we set the sid of the ghost task to be _the same_ as that of the parent task (and not the pid of the parent task itself - which only works when the parent is indeed the session leader).

2) Consider a checkpoint with 2 tasks: one has a pgid corresponding to a dead process, the other has its sid corresponding to the same dead process. If the former is processed by ckpt_setup_task() first, it will produce a ghost task that is _not_ a session leader - which is incorrect. Moreover, this will not be recrtified later when the latter task is found.

We fix this by splitting the work into two loops: in the first we only consider tasks' sids and add the corresponding ghosts. This ensures that they are session leaders. In the second we consider the dead pgids and the tgids.

The patch also refactors ckpt_set_task() into smaller pieces to simplify its logic and improve readability.

Signed-off-by: Oren Laadan <orenl@cs.columbia.edu>

```
---
 restart.c |   83 +++++++++++++++++++++++++++++++++++++++++++++++++++++--------------
 1 files changed, 63 insertions(+), 20 deletions(-)

diff --git a/restart.c b/restart.c
index f64e508..e60335c 100644
--- a/restart.c
+++ b/restart.c
@@ -1456,22 +1456,12 @@ static int ckpt_build_tree(struct ckpt_ctx *ctx)
  return 0;
 }

-static int ckpt_setup_task(struct ckpt_ctx *ctx, int pid_ind, int ppid_ind)
+static int ckpt_ghost_task(struct ckpt_ctx *ctx, struct ckpt_pids *pids,
```

```
+      int pid_ind, int sid_ind, int ppid_ind)
 {
  struct task *task;
- struct ckpt_pids *pids;
  int j;

- /* ignore if outside namespace */
- if (pid_ind == 0 || pid_ind == CKPT_PID_ROOT)
-  return 0;
-
- pids = pids_of_index(ctx, pid_ind);
-
- /* skip if already handled */
- if (hash_lookup(ctx, pids->numbers[0]))
-  return 0;
-
  task = &ctx->tasks[ctx->tasks_cnt++];

  task->flags = TASK_GHOST;
@@ -1479,7 +1469,7 @@ static int ckpt_setup_task(struct ckpt_ctx *ctx, int pid_ind, int ppid_ind)
  task->pid_ind = pid_ind;
  task->ppid_ind = ppid_ind;
  task->tgid_ind = pid_ind;
- task->sid_ind = ppid_ind;
+ task->sid_ind = sid_ind;

  task->children = NULL;
  task->next_sib = NULL;
@@ -1500,6 +1490,47 @@ static int ckpt_setup_task(struct ckpt_ctx *ctx, int pid_ind, int
ppid_ind)
  return 0;
 }

+static struct ckpt_pids *ckpt_consider_pid(struct ckpt_ctx *ctx, int pid_ind)
+{
+ struct ckpt_pids *pids;
+
+ /* ignore if outside namespace */
+ if (pid_ind == 0 || pid_ind == CKPT_PID_ROOT)
+  return NULL;
+
+ pids = pids_of_index(ctx, pid_ind);
+
+ /* skip if already handled */
+ if (hash_lookup(ctx, pids->numbers[0]))
+  return NULL;
+
+ return pids;
```

```
+}
+
+static int ckpt_set_session(struct ckpt_ctx *ctx, int pid_ind, int ppid_ind)
+{
+ struct ckpt_pids *pids;
+
+ pids = ckpt_consider_pid(ctx, pid_ind);
+ if (!pids)
+  return 0;
+
+ return ckpt_ghost_task(ctx, pids, pid_ind, pid_ind, ppid_ind);
+}
+
+static int ckpt_set_task(struct ckpt_ctx *ctx, int pid_ind, int ppid_ind)
+{
+ struct ckpt_pids *pids;
+ struct task *parent;
+
+ pids = ckpt_consider_pid(ctx, pid_ind);
+ if (!pids)
+  return 0;
+
+ parent = hash_lookup_ind(ctx, ppid_ind);
+ return ckpt_ghost_task(ctx, pids, pid_ind, parent->sid_ind, ppid_ind);
+}
+
 static int _ckpt_valid_pid(struct ckpt_ctx *ctx, pid_t pid, char *which, int i)
 {
  if (pid < 0) {
@@ -1697,17 +1728,29 @@ static int ckpt_init_tree(struct ckpt_ctx *ctx)

  ctx->tasks_cnt = ctx->tasks_nr;

- /* add pids unaccounted for (no tasks) */
+ /*
+  * Add sids unaccounted for (no tasks): find those orphan pids
+  * that are used as some task's sid, and create ghost session
+  * leaders for them. Since they should be session leaders, we
+  * must do this before adding other ghost tasks for tgid/pgid
+  * below: the latter is added a non-session leader.
+  */
  for (i = 0; i < ctx->tasks_nr; i++) {
-
   /*
    * An unaccounted-for sid belongs to a task that was a
-  * session leader and died. We can safe set its parent
+  * session leader and died. We can safely set its parent
    * (and creator) to be the root task.
```

```
    */
-   if (ckpt_setup_task(ctx, tasks_arr[i].vsid, root_pid_ind) < 0)
+
+   if (ckpt_set_session(ctx, tasks_arr[i].vsid, root_pid_ind) < 0)
        return -1;
+ }

+ /*
+  * Add tgids/pgids unaccounted for (no tasks): find those
+  * orphan tgids/pgids and create ghost tasks for them.
+  */
+ for (i = 0; i < ctx->tasks_nr; i++) {
    /*
     * An sid == 0 means that the session was inherited an
     * ancestor of root_task, and more specifically, via
@@ -1723,18 +1766,18 @@ static int ckpt_init_tree(struct ckpt_ctx *ctx)
     * need to add it with the same sid as current (and
     * other) threads.
     */
-   if (ckpt_setup_task(ctx, tasks_arr[i].vtgid, ppid_ind) < 0)
+   if (ckpt_set_task(ctx, tasks_arr[i].vtgid, ppid_ind) < 0)
        return -1;

    /*
     * If pgrp == sid, then the pgrp/sid will already have
     * been hashed by now (e.g. by the call above) and the
-    * ckpt_setup_task() will return promptly.
+    * ckpt_set_task() will return promptly.
     * If pgrp != sid, then the pgrp 'owner' must have the
     * same sid as us: all tasks with same pgrp must have
     * their sid matching.
     */
-   if (ckpt_setup_task(ctx, tasks_arr[i].vpgid, ppid_ind) < 0)
+   if (ckpt_set_task(ctx, tasks_arr[i].vpgid, ppid_ind) < 0)
        return -1;
  }

--
1.7.1
```

_____

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