Subject: Re: [PATCH 4/5] c/r: checkpoint and restart pids objects
Posted by Oren Laadan on Sat, 05 Feb 2011 22:21:43 GMT
View Forum Message <> Reply to Message

Suka,

Thanks for the review.

On 02/05/2011 04:43 PM, Sukadev Bhattiprolu wrote:
> Oren:
>
> I am still reviewing this patchset, but have a few questions/comments
> below on this patch.
>
> | From: Oren Laadan <orenl@cs.columbia.edu>
> | Subject: [PATCH 4/5] c/r: checkpoint and restart pids objects
> |
> | Make use of (shared) pids objects instead of simply saving the pid_t
> | numbers in both checkpoint and restart.
> |
> | The motivation for this change is twofold. First, since pid-ns came to
> | life pid's in the kenrel _are_ shared objects and should be treated as
> | such. This is useful e.g. for tty handling and also file-ownership
> | (the latter waiting for this feature). Second, to properly support
> | nested namesapces we need to report with each pid the entire list of
> | pid numbers, not only a single pid. While current we do that for all
> | "live" pids (those that belong to live tasks), we didn't do it for
> | "dead" pids (to be assigned to ghost restarting tasks).
> |
> | Note, that ideally the list of vpids of a pid object should also
> | include the pid-ns to which each level belongs; however, in this patch
> | we don't yet hanlde that. So only linear pid-nesting works well and
> | not arbitrary tree.
> |
> | DICLAIMER: this patch is big and intrusive! Here is a summary of the
> | changes that it makes:

[...]

> | diff --git a/include/linux/checkpoint_hdr.h b/include/linux/checkpoint_hdr.h
> | index 922eff0..c0a548a 100644
> | --- a/include/linux/checkpoint_hdr.h
> | +++ b/include/linux/checkpoint_hdr.h
> | @@ -107,7 +107,9 @@ enum {
> | CKPT_HDR_SECURITY,
> | #define CKPT_HDR_SECURITY CKPT_HDR_SECURITY
> |
> | - CKPT_HDR_TREE = 101,

> | + CKPT_HDR_PIDS = 101,
> | +#define CKPT_HDR_PIDS CKPT_HDR_PIDS
> | + CKPT_HDR_TREE,
> | #define CKPT_HDR_TREE CKPT_HDR_TREE
> | CKPT_HDR_TASK,
> | #define CKPT_HDR_TASK CKPT_HDR_TASK
> | @@ -358,20 +360,32 @@ struct ckpt_hdr_container {
> | */
> | } __attribute__((aligned(8)));;
> |
> | +/* pids array */
> | +struct ckpt_hdr_pids {
> | + struct ckpt_hdr h;
> | + __u32 nr_pids;
> | + __u32 nr_vpids;
> | +} __attribute__((aligned(8)));
>
> For consistency can we call this ckpt_hdr_pids_tree ?

'struct ckpt_hdr_pids' and 'struct ckpt_pids' are related, and
do not provide information about the process tree. See also,
for example, 'struct ckpt_eventpoll_item' and the .._hdr_.. one.

>
> | +
> | +struct ckpt_pids {
> | + __u32 depth;
> | + __s32 numbers[1];
> | +} __attribute__((aligned(8)));
> | +
>
> This actually corresponds to _one_ 'struct pid' right ? Can we rename to
> 'struct ckpt_pid' or ckpt_struct_pid ?

Yes, it corresponds to a single pid object, which has _multiple_
(rather than a single) pids numbers --> hence the name.

>
> | /* task tree */
> | struct ckpt_hdr_tree {
> | struct ckpt_hdr h;
> | - __s32 nr_tasks;
> | + __u32 nr_tasks;
> | } __attribute__((aligned(8)));
>
> And this to, ckpt_hdr_task_tree ?

Ok.

[...]

> | + for (n = 0; n < ctx->nr_tasks; n++) {
> | + task = ctx->tasks_arr[n];
> | +
> | + rcu_read_lock();
> | + pid = get_pid(task_pid(task));
> | + tgid = get_pid(task_tgid(task));
> | + pgrp = get_pid(task_pgrp(task));
> | + session = get_pid(task_session(task));
> | + rcu_read_unlock();
> | +
> | + /*
> | +   * How to handle references to pids outside our pid-ns ?
> | +   * In container checkpoint, such pids are prohibited, so
> | +   * we report an error.
> | +   * In subtree checkpoint it is valid, however, we don't
> | +   * collect them here to not leak data (it is irrelevant
> | +   * to userspace anyway), Instead, in checkpoint_tree() we
> | +   * substitute 0 for the such pgrp/session entries.
> | +   */
> | +
> | + /* pid */
> | + ret = ckpt_obj_lookup_add(ctx, pid,
> | +   CKPT_OBJ_PID, &new);
> | + if (ret >= 0 && new) {
> | +   depth += pid->level - root_pidns->level;
>
> 'depth' here was a bit confusing to me. We are really counting of the
> number of vpids. So, can you rename 'depth' to nr_pids ?

So either 'vpids', or 'levels'. The problem with 'nr_pids' is that
it's ambiguous: could be number of pid-objects, or pid-numbers.

>
> (i.e if you find a process with pid and tgid two levels deep, it initially
> appeared that the depth would be 4. But the depth is still 2 and the number
> of vpids is 4 right ?)

Yes, it is summing the depths.

> | +   ret = flex_array_put(pids_arr, i++, pid, GFP_KERNEL);
> | +   new = 0;
> | + }
> | +
> | + /* tgid: if tgid != pid) */
> | + if (ret >= 0 && tgid != pid)

```
> | +   ret = ckpt_obj_lookup_add(ctx, tgid,
> | +         CKPT_OBJ_PID, &new);
> | +  if (ret >= 0 && new) {
> | +   depth += tgid->level - root_pidns->level;
> | +   ret = flex_array_put(pids_arr, i++, tgid, GFP_KERNEL);
> | +   new = 0;
> | +  }
> | +
> | +  /*
> | +   * pgrp: if in our pid-namespace, and
> | +   *       if pgrp != tgid, and if pgrp != root_session
> | +   */
> | +  if (pid_nr_ns(pgrp, root_pidns) == 0) {
> | +   /* pgrp must be ours in container checkpoint */
> | +   if (!(ctx->uflags & CHECKPOINT_SUBTREE))
> | +    ret = -EBUSY;
> | +  } else if (ret >= 0 && pgrp != tgid && pgrp != root_session)
> | +   ret = ckpt_obj_lookup_add(ctx, pgrp,
> | +         CKPT_OBJ_PID, &new);
> | +  if (ret >= 0 && new) {
> | +   depth += pgrp->level - root_pidns->level;
> | +   ret = flex_array_put(pids_arr, i++, pgrp, GFP_KERNEL);
> | +   new = 0;
> | +  }
> | +
> | +  /*
> | +   * session: if in our pid-namespace, and
> | +   *          if session != tgid, and if session != root_session
> | +   */
> | +  if (pid_nr_ns(session, root_pidns) == 0) {
> | +   /* session must be ours in container checkpoint */
> | +   if (!(ctx->uflags & CHECKPOINT_SUBTREE))
> | +    ret = -EBUSY;
> | +  } else if (ret >= 0 && session != tgid && session != root_session)
> | +   ret = ckpt_obj_lookup_add(ctx, session,
> | +         CKPT_OBJ_PID, &new);
> | +  if (ret >= 0 && new) {
> | +   depth += session->level - root_pidns->level;
> | +   ret = flex_array_put(pids_arr, i++, session, GFP_KERNEL);
> | +  }
> | +
> | +  put_pid(pid);
> | +  put_pid(tgid);
> | +  put_pid(pgrp);
> | +  put_pid(session);
>
> We save the pid pointers in the flex_array right ? If we put the references
> here, the pointers in flex_array don't have a reference, so the pid pointer
```

> access in checkpoint_pids_dump() is unsafe ?
>
> Or is it that the process tree is frozen so the pid won't go away ? If
> so do we need the get_pid() and put_pid() in this function ?

We get a reference inside the rcu_read_lock() so that we could safely
access them after we drop the lock. Then we add each (new) pid to the
objhash - which will take another reference to it. Finally we drop
the local reference no longer needed.

I'll a comment to make it clear.

> | +
> | + if (ret < 0)
> | +   break;
> | + }
> | +
> | + *nr_pids = i;
> | + *nr_vpids = depth;
> | +
> | + ckpt_debug("nr_pids = %d, nr_vpids = %d\n", i, depth);
> | + return ret;
> | +}
> | +
> | +static int checkpoint_pids_dump(struct ckpt_ctx *ctx,
> | +   struct flex_array *pids_arr,
> | +   int nr_pids, int nr_vpids)
> | +{
> | + struct ckpt_hdr_pids *hh;
> | + struct ckpt_pids *h;
> | + struct pid *pid;
> | + char *buf;
> | + int root_level;
> | + int len, pos;
> | + int depth = 0;
>
> Here too, using 'depth' to count nr_vpids is a bit confusing :-)

Ok - will change as above.

>
> | + int i, n = 0;
> | + int ret;
> | +
> | + hh = ckpt_hdr_get_type(ctx, sizeof(*hh), CKPT_HDR_PIDS);
> | + if (!hh)
> | + return -ENOMEM;
> | +

> | + hh->nr_pids = nr_pids;
> | + hh->nr_vpids = nr_vpids;
> | +
> | + ret = ckpt_write_obj(ctx, &hh->h);
> | + ckpt_hdr_put(ctx, hh);
> | + if (ret < 0)
> | + return ret;
> | +
> | + pos = (nr_pids * sizeof(*h)) + (nr_vpids * sizeof(__s32));
> | + ret = ckpt_write_obj_type(ctx, NULL, pos, CKPT_HDR_BUFFER);
> | + if (ret < 0)
> | + return ret;
> | +
> | + buf = ckpt_hdr_get(ctx, PAGE_SIZE);
> | + if (!buf)
> | + return -ENOMEM;
> | +
> | + root_level = ctx->root_nsproxy->pid_ns->level;
> | +
> | + while (n < nr_pids) {
> | + pos = 0;
> | +
> | + rcu_read_lock();
> | + while (1) {
> | +   pid = flex_array_get(pids_arr, n);
> | +   len = sizeof(*h) + pid->level * sizeof(__s32);
>
> Hmm. pid->level is the global level here right ? So if we checkpoint a
> container 2 levels deep, we don't need to save the vpids for levels 0,1.
> do we ? Or should we s/pid->level/(pid->level - root->level)/ (like
> we do for h->depth below ?

The latter. Good catch !

>
> | +
> | +   /* need to flush current buffer ? */
> | +   if (pos + len > PAGE_SIZE || n == nr_pids)
> | +   break;
> | +
> | +   h = (struct ckpt_pids *) &buf[pos];
> | +   h->depth = pid->level - root_level;
> | +   for (i = 0; i <= h->depth; i++)
> | +   h->numbers[i] = pid->numbers[pid->level + i].nr;
> | +   depth += h->depth;
> | +   pos += len;
> | +   n++;
> | + }

> | + rcu_read_unlock();
> | +
> | + /* something must have changed since last count... */
> | + if (depth > nr_vpids) {
> | +   ret = -EBUSY;
> | +   break;
> | + }
> | +
> | + ret = ckpt_kwrite(ctx, buf, pos);
> | + if (ret < 0)
> | +   break;
>
> Do we need to memset(buf, 0, sizeof(buf)) here ? Specially if we expect
> to fill 0s in ancestor pid namespaces (in the above example of
> checkpointing a container 2 levels deep, do we want to write zeros for
> the pid in levels 0,1) ?

We shouldn't need it - assuming we fix the above as noted.

Thanks,

Oren.

_____

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