Virtuozzo

Checkpoint-restoring containers with Docker inside

Alexander Mikhalitsyn
Software Developer at Virtuozzo

Pavel Tikhomirov
Software developer at Virtuozzo

Aug 2020

Agenda

- Nested PID namespaces
- Several UTS/IPC namespaces
- Nested USER namespaces
- Internal overlayfs mounts

LINUX
PLUMBERS CONFERENCE / August 24-28 2020

Virtuozzo

General namespace C/R design

- Namespaces are stored in global single linked list ns_ids (struct ns_id).
- Each ns_id has struct ns_desc - namespace descriptor (clone flag, “string” name in /proc/pid/ns/<..>)
- nsid_add() function adds new namespace to list
- rst_add_ns_id() is wrapper around nsid_add() to add new ns_id on **restore** stage
- __get_ns_id() is wrapper of nsid_add() on **dump** stage

General namespace C/R design (OLD)

Dump

- cr_dump_tasks
  -> collect_pstree_ids
  -> loop get_task_ids
  -> dump_task_ns_ids
  -> __get_ns_id
- We have no separate image for ns_id info

Restore

- cr_restore_tasks
  -> prepare_pstree
  -> read_pstree_image
  -> loop read_one_pstree_item
  -> read_pstree_ids
  -> rst_add_ns_id

- Entering to namespace in restore_one_alive_task or earlier

General namespace C/R design (NEW)

Dump

- cr_dump_tasks
  -> collect_pstree_ids
  -> loop get_task_ids
  -> dump_task_ns_ids
  -> __get_ns_id
- **We have separate image and can store parent nsid, userns id.**
- **collect_ns_hierarhy**
  -> loop set_ns_hookups
  -> set_ns_opt (parent, userns)

Restore

- **cr_restore_tasks**
  -> read_ns_with_hookups
  -> rst_new_ns_id
- Entering to namespace in restore_one_alive_task or earlier

Nested USER namespaces

- We can do setns and go to the USER namespace immediately
- USER namespaces can be nested
- So, we can reconstruct hierarchy of user namespaces independently from processes tree and then do setns() on each process
- But we can’t do setns to higher USER ns from lower-level

Nested USER namespaces

Dump

- cr_dump_tasks
  -> collect_pstree_ids
  -> loop get_task_ids
  -> dump_task_ns_ids
  -> __get_ns_id

- **collect_user_namespaces**
  -> loop collect_user_ns
  -> __dump_user_ns

- collect_ns_hierarhy
  -> loop set_ns_hookups
  -> set_ns_opt (parent, userns)

Restore

- **prepare_namespace**
  -> create_user_ns_hierarhy
  -> prepare_userns

- restore_one_alive_task
  -> set_user_ns
  -> setns

Nested PID namespaces (1)

- We can’t do setns and go to the PID namespace immediately
- PID namespaces can be nested
- So during restoring pstree we call clone with CLONE_NEWPID flag to create and get into correct pidns from the birth of the process
- Will call setns to restore pidns for children

Nested PID namespaces (2)

Dump

- cr_dump_tasks
  -> collect_pstree_ids
  -> loop get_task_ids
  -> dump_task_ns_ids
  -> __get_ns_id

- collect_ns_hierarhy
  -> loop set_ns_hookups
  -> set_ns_opt (parent, userns)

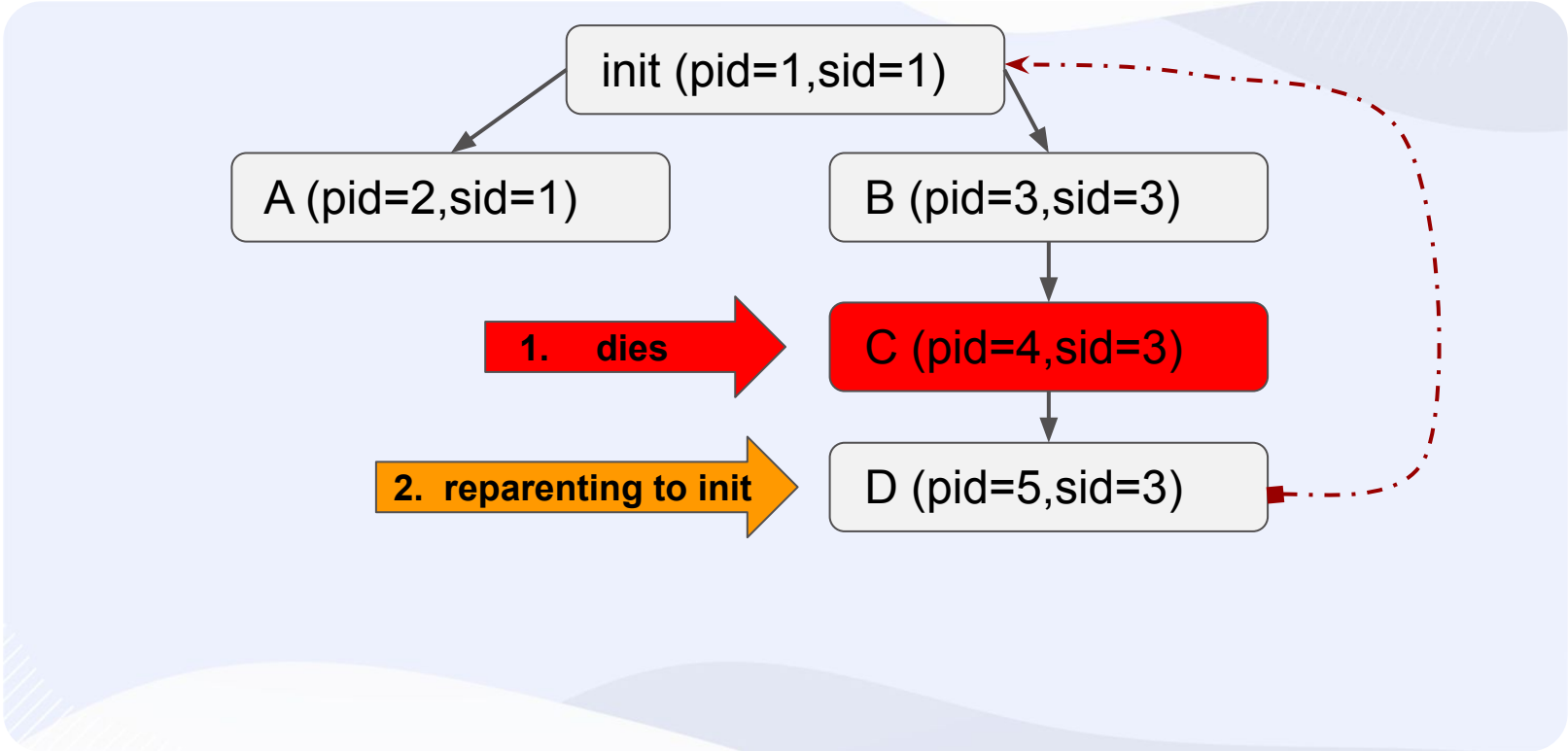
Restore

- restore_task_with_children
  -> setup_current_pid_ns
  -> create_pid_ns_helper
- restore_one_alive_task
  restore_task_pfc_before_user_ns
  set_pid_for_children_ns
- restore_one_alive_task
  sigreturn_restore
  __export_restore_task
  __export_restore_thread
  setns(CLONE_NEWPID)
- destroy_pid_ns_helpers
  -> loop SIGKILL

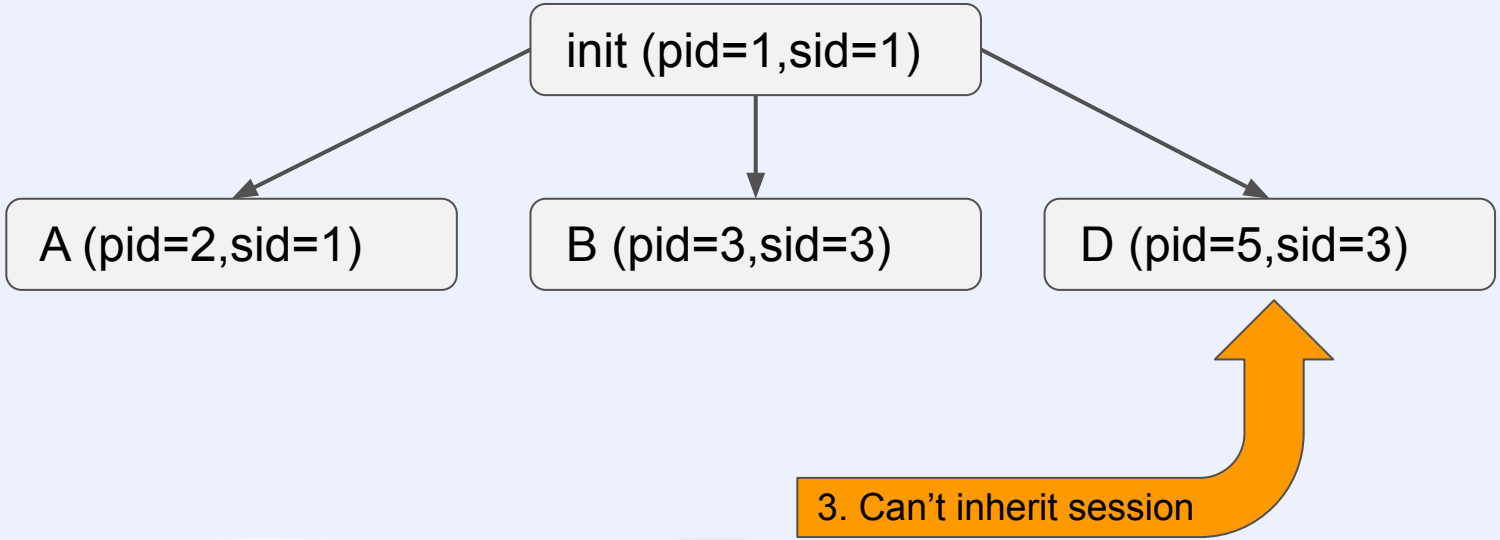
Nested PID namespaces (3)

- Process groups (we can setpgid(pid) - **ok**)
- Sessions (inherit) (we can only create new session and inherit - **problem**)
- **Sessions restore when some tasks were reparented to init**
- **Process trees where some process (in one session) has set PR_SET_CHILD_SUBREAPER and reparenting occurred for the process from another session**

Nested PID namespaces (sessions 1)



Nested PID namespaces (sessions 2)



Several UTS/IPC namespaces

Dump

- collect_pstree_ids
  -> for_each_pstree_item
  -> dump_task_ns_ids

  … dump_pstree(root_task)...

- dump_namespaces(root_task)
  -> dump_uts_ns (or ipc, or net)

Restore

- restore_task_with_children
  -> make_root_ns(uts)

- restore_task_with_children
  -> prepare_namespace(root_task)
  -> prepare_namespaces(uts)

- restore_one_alive_task
  -> restore_task_ns (do setns)

Internal overlayfs mounts (1)

- Overlayfs mounts have several path options: lowerdir, workdir, upperdir
- Example:
  overlay on /var/lib/docker/overlay2/XYZ/merged type overlay (<...>,**lowerdir**=/var/lib/docker/overlay2/XYZ-init/diff**:**/var/lib/docker/overlay2/ABC/diff,**upperdir**=/var/lib/docker/overlay2/XYZ/diff,**workdir**=/var/lib/docker /overlay2/XYZ/work)
- But also overlayfs could be mounted with relative paths in these options

Internal overlayfs mounts (2)

- After pivot_root syscall paths won't be recalculated!
  overlay on /var/lib/docker/overlay2/XYZ/merged type overlay (<...>,**lowerdir=/tmp/mntns.XXXX./yard**/var/lib/docker/overlay2/XYZ-init/diff**:/tmp/mntns.XXXX./yard**/var/lib/docker/overlay2/ABC/diff,**upperdir=/tmp/mntns.XXXX./yard**/var/lib/docker/overlay2/XYZ/diff,**workdir=/tmp/mntns.XXXX./yard**/var/lib/docker/overlay2/XYZ/work)
- We don't know from which mount these paths were opened

Links

1. Virtuozzo CRIU sources https://src.openvz.org/projects/OVZ/repos/criu
2. Overlayfs support in CRIU https://src.openvz.org/projects/OVZ/repos/criu/commits/8ee9695522f4f98a21ef7fad23d0a9ccc72ecdc6
3. Namespaces connections ioctls https://lwn.net/Articles/698364/
4. UTS/IPC namespaces https://src.openvz.org/projects/OVZ/repos/criu/commits/705e08f9d8bb40cf455296bcb987d2f26cd39401
5. Overlayfs kernel patch https://lore.kernel.org/lkml/20200604161133.20949-1-alexander.mikhalitsyn@virtuozzo.com/
6. nsfs: Introduce ioctl to set vector of ns_last_pid's on pid ns hierarhy https://lore.kernel.org/lkml/149245014695.17600.12640895883798122726.stgit@localhost.localdomain/

Virtuozzo

LINUX
PLUMBERS CONFERENCE

Thank you!