CRIU and the PID dance

Adrian Reber
Red Hat

2019

This article is the basis for the presentation "CRIU and the PID dance" at the Linux Plumbers Conference 2019. This article describes the current steps to restore a process with a certain PID using CRIU, which is called the "PID dance". The reasons for the PID dance are mentioned and why it would be good to avoid it with an extension to the newly introduced system call "clone3()". Reasons to relax the current requirement of CAP_SYS_ADMIN are discussed which would enable rootless container migration as well as rootless checkpoint and restore of MPI based applications in HPC environments.

I. Introduction

To correctly restore processes Checkpoint/Restore In Userspace (CRIU)[1] has to restore the processes with the same process identifiers (PID). One of the main reasons for the restored processes to have the same PIDs is to ensure that the parent child relations of the restored process tree are the same.

To restore a process with the same PID it had during checkpointing, CRIU does its PID dance: `open()`, `write()`, `close()`, `clone()` and `getpid()`.

The PID dance is slow as it requires multiple system calls and it opens up the possibility for a race condition. Between setting the desired PID via `open()`, `write()` and `close()` and the actual `clone()` another process could have been created and CRIU's `clone()` will get the wrong PID.

Therefore a new interface to create processes with a certain PID has been introduced and will be described here.

This article is the basis for the presentation "CRIU and the PID dance" at the Linux Plumbers Conference 2019[2].

II. CRIU Details

CRIU was first discussed in 2011 at the Linux Plumbers Conference[3] and has since been integrated in multiple container runtimes to support container migration.

One of the goals of CRIU is to be as transparent as possible, which means that CRIU does not require the processes to be prepared or instrumentalized in any way. The goal of being as transparent as possible also means that the processes have to be restored with the same PID as the processes had during checkpointing. For single processes this is usually not important but as CRIU always checkpoints and restores complete process trees it is important that the parent child relations of all processes is maintained.

One of the drawbacks of restoring all processes with the same PID as during checkpointing is that it opens CRIU up to PID collisions. If a process with the PID of the to be restored process already exists, the restore will fail and CRIU will abort.

PID namespaces are a possibility to avoid PID collisions and especially in the container migration use case it is unlikely that CRIU fails due to a PID collision. It is, however, important to remember that PID namespaces are not required and a restore can always happen in the host PID namespace.

To restore a process CRIU morphs itself into the to be restored process. For a process tree this means that CRIU forks multiple times to re-create the process tree in the same form it had during checkpointing. Once all the necessary processes have been created these processes are transformed into the to be restored processes. To restore the processes with the same PID as the processes had during checkpointing CRIU currently has to do the PID dance.

The PID dance consists of the following steps:

- `open()` `/proc/sys/kernel/ns_last_pid`[4]
- `write()` desired (PID - 1) to `ns_last_pid`
- `close()` `ns_last_pid`
- `clone()`
- `getpid()` to verify if the PID of the newly created process actually matches the desired PID. If another PID is returned CRIU will abort at this point.

The PID dance for child processes happens relatively early in the restore process, for threads it happens relatively late.

The problem with this PID dance is that it is slow as it requires multiple system calls and it is open to race conditions. It can always happen that between setting the desired PID via `ns_last_pid` and the actual `clone()` another process, independent of the restore, is created, which means that `getpid()` will not return the desired PID and CRIU will abort.

Using PID namespaces the race condition becomes unlikely but it still exists. PID namespaces do not help with the fact that the dance requires multiple system calls.

One important point for further discussions is that currently `ns_last_pid` requires the capability `CAP_SYS_ADMIN`.

III. `CLONE3()` WITH `SET_TID`

With the introduction of CLONE_PIDFD[5] the last available flag for `clone()` was used and if further flags should be passed to `clone()` a new interface was necessary. Fortunately a new interface was included in the kernel version 5.3: `clone3()`[6]

In its commit message `clone3()` has the following description:

> ”In general, clone3() is extensible and allows for the implementation of new features.”[6]

When I first heard about `clone3()` I immediately thought about extending it to make it possible to `clone()` with a certain PID. Creating new processes with a certain PID has been something that also other checkpoint/restore implementations tried to solve before CRIU existed in 2010: `eclone()`[7]

A first patch to extend `clone3()` was shared between a small group of people, but the feedback was not very positive. Compared to the `ns_last_pid` approach my first patch opened up `clone3()` with `set_tid` for very user. No `CAP_SYS_ADMIN` was required, every user was able to create processes with specifying a desired PID independent of the PID namespace. To implement `clone3()` with `set_tid` without any restrictions was motivated by making it easier to restore processes as non-root for rootless containers or for checkpointing and restoring MPI processes.

As the feedback was not very positive and I myself was unsure how important it actually is for CRIU to have `clone3()` with `set_tid` I did not continue with this patch.

A few weeks later I started to discuss `clone3()` with `set_tid` with a few of the CRIU developers and the feedback was that something like `clone3()` with `set_tid` would be important for CRIU because of the currently necessary PID dance. With the PID dance there is always the possibility for race conditions and it is slow.

The largest difference to my first patch was that this time the implementation of `clone3()` with `set_tid` required the same capabilities as the `ns_last_pid` approach. This patch solved the race condition and reduced the number of system calls CRIU would need to create a process with a certain PID, this patch did not change the required capabilities.

After a few rounds of feedback the patch was accepted by everyone involved in the discussion and if it accepted during the next merge window CRIU can start to remove the PID dance when running on the latest version of the kernel.

By setting `set_tid` of `struct clone_args` to the desired PID it is now possible to create a process with `clone3()` which has the PID specified in `set_tid`. It has the usual restrictions that it fails if the PID is already in use, a PID namespace always has to have a PID 1 and it still requires `CAP_SYS_ADMIN`.

IV. Relax `CAP_SYS_ADMIN`

With the patch to add `set_tid` to `clone3()` it is now possible to restore processes without the described race condition and faster as it is using less system calls. The next step would be to discuss if the requirement of `CAP_SYS_ADMIN` could be relaxed. For the `ns_last_pid` approach as well as for the `clone3()` approach.

Reasons to use something else than `CAP_SYS_ADMIN` is to make it possible to checkpoint and restore rootless container thus enabling rootless container migration. Using another capability could also enable checkpointing and restoring of MPI processes in HPC environments without requiring root.

One possible idea is to introduce the new capability `CAP_RESTORE` as proposed in the upcoming Linux Plumbers Conference 2019[8].

References

[1] *Checkpoint/Restore In Userspace (CRIU).* [Online; accessed 2019-05-03]. URL: `https://criu.org/`.

[2] *CRIU and the PID dance.* [Online; accessed 2019-08-27]. URL: `https://linuxplumbersconf.org/event/4/contributions/472/`.

[3] *Checkpoint/restart in the userspace*. [Online; accessed 2019-05-03]. URL: http://blog.linuxplumbersconf.org/2011/ocw/sessions/831.

[4] *ns_last_pid*. [Online; accessed 2019-08-23]. URL: https://www.kernel.org/doc/Documentation/sysctl/kernel.txt.

[5] *clone: add CLONE_PIDFD*. [Online; accessed 2019-08-26]. URL: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=b3e5838252665ee4c.

[6] *fork: add clone3*. [Online; accessed 2019-08-26]. URL: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=7f192e3cd316ba58c.

[7] *c/r: introduce checkpoint/restore methods to struct proto_ops*. [Online; accessed 2019-08-26]. URL: https://lore.kernel.org/patchwork/patch/198220/.

[8] *Update on Task Migration at Google Using CRIU*. [Online; accessed 2019-08-27]. URL: https://linuxplumbersconf.org/event/4/contributions/508/.