Memory tracking for iterative container migration

Mike Rapoport
<rppt@linux.vnet.ibm.com>

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements No 645402 and No 688386

Current state

- Using soft-dirty
- Iterations are independent
- Control is outside CRIU scope

Userfaultfd-WP

- Userfaultfd notification for WRITE page faults
- More flexible and robust than soft-dirty
- May obsolete soft-drity

Possible flow

Iteration 1

- Start memory tracker daemon
- Freeze tasks
- Create memory pre-dump
- Register task memory with uffd
  - Pass the uffd to the daemon
- Un-freeze tasks
- The tracker monitors page writes

Iteration 2..n-1

- Freeze tasks
- Get dirty pages bitmap
- Dump dirty pages
- Un-freeze tasks

Possible flow (cont)

Iteration 2..n-1

- Freeze tasks
- Get dirty pages bitmap?
- Dump dirty pages
- Un-freeze tasks

Possible flow (cont again)

Iteration n

- Freeze tasks
- Get dirty pages bitmap
- Dump dirty pages
- Unregister uffd
- Complete dump

Memory tracker

- Receive uffds from the dump
- Process WRITE faults
- Process bitmap requests

Open points

- Who is responsible for saving modified pages
  - Memory tracker vs dump
- How memory tracker and dump communicate
  - UNIX socket? Something else?
- Where and how control should be implemented
  - P.Haul, container engines, both?

References

https://www.kernel.org/doc/Documentation/vm/userfaultfd.txt

http://man7.org/linux/man-pages/man2/userfaultfd.2.html

https://schd.ws/hosted_files/lcccna2016/c4/userfaultfd.pdf

http://wiki.qemu.org/Features/PostCopyLiveMigration

https://criu.org/Lazy_migration

https://medium.com/@MartinCracauer/generational-garbage-collection-write-barriers-write-protection-and-userfaultfd-2-8b0e796b8f7f

Thank you!