Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

Unsolved CRIU problems

Pavel Tikhomirov <snorcht@gmail.com>
Andrei Vagin <avagin@gmail.com>

LINUX PLUMBERS CONFERENCE | Vienna, Austria
Sept. 18-20, 2024

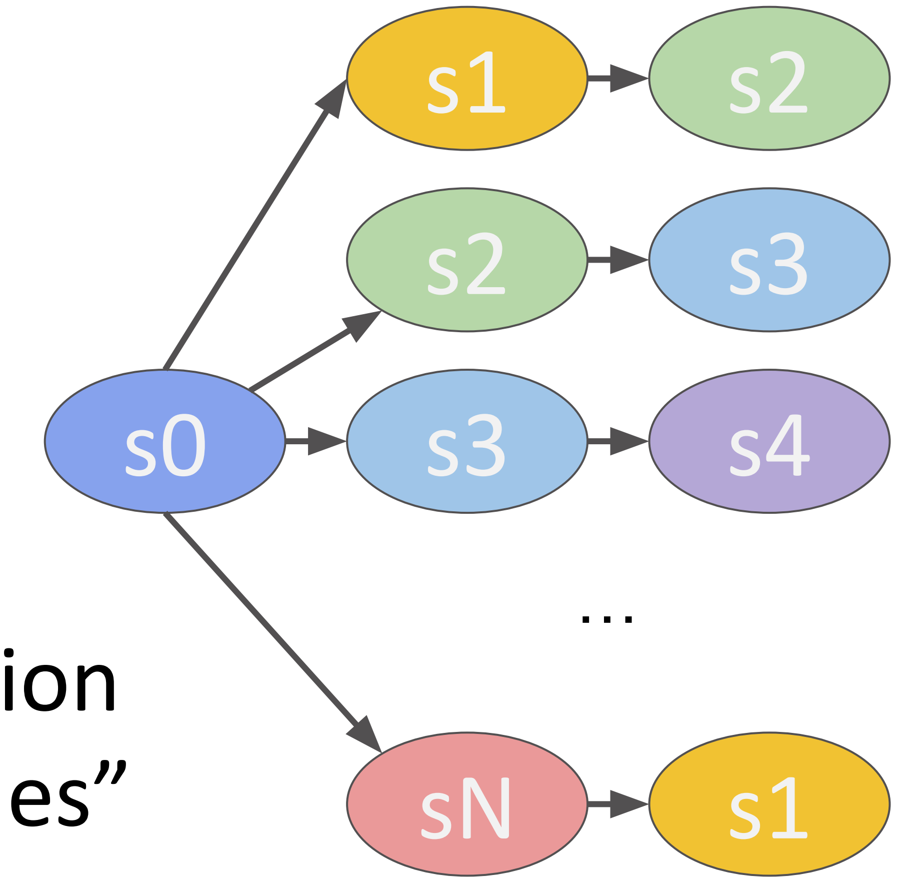
Agenda

- Restoring complex process trees (sessions)
- Nested pid and user namespaces restore using clone3 + set_tid syscall
- Migration across mismatching CPUs
- Dumping COW-ed memory effectively

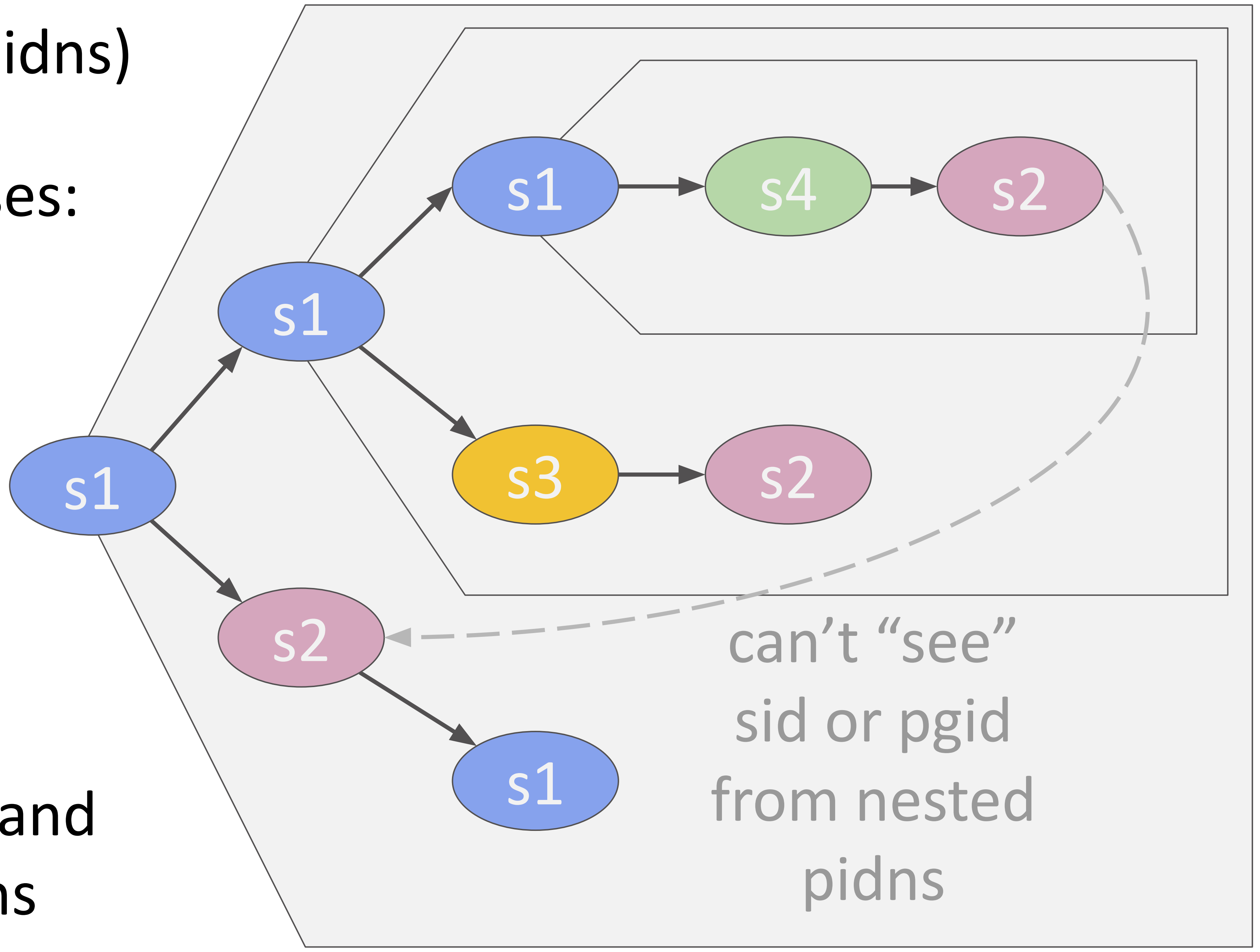
Restoring complex process trees (sessions)

- Session can only be inherited and only be created by specific process
- Session leader can have “old session” in descendants
- Reparenting on parent exit to a pidns init (of parent) or to a child subreaper
- Clone with CLONE_PARENT gives sibling session
- Nested pidns-es with setns
- Can’t setpgid to process group “0” (from nested pidns)

Really hard to figure out how to restore complex cases:



session “cycles”



pidns and setns

Restoring complex process trees (sessions)

Possible solution:

Get rid of inherit-only resources in Linux Kernel:

- new syscall to allow attaching to pre-existing sessions and process groups
- make it pidfd based to overcome nested pidns pid visibility problem

pros:

- don’t need to invent complex order of syscalls to recreate original state in CRIU

cons:

- connecting to foreground group may allow us read sensitive data from controlling terminal (e.g. we can only allow attaching to a session without a controlling terminal?)

Ideas, thoughts, any security problems?
Do we have or plan to have any other inherit-only kernel resources?

Restoring complex process trees (sessions)

“Backup” solution:

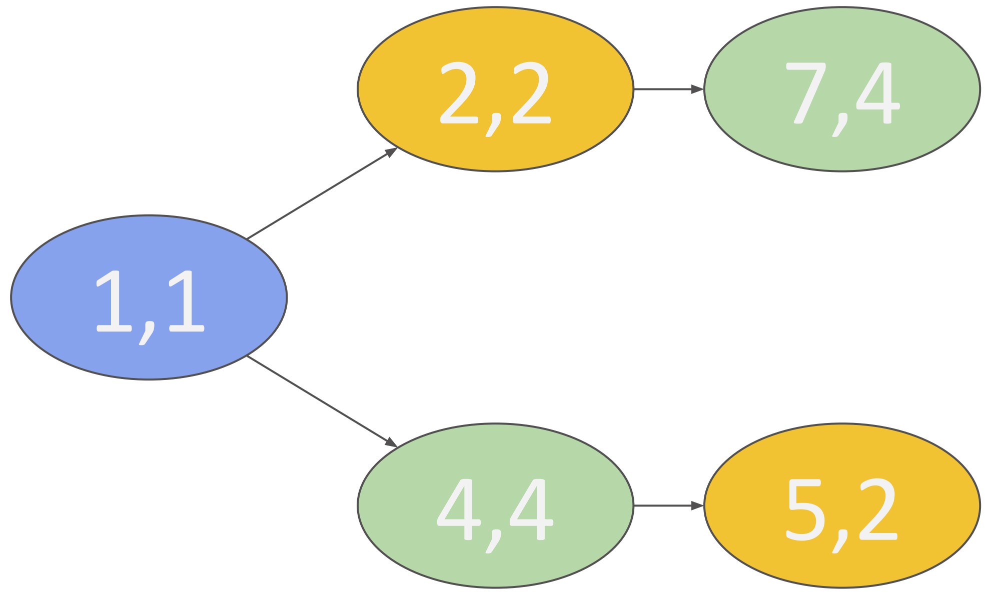
CABA (closest alive born ancestor) auxiliary tree

- remembers all alive ancestors from “historical” process tree

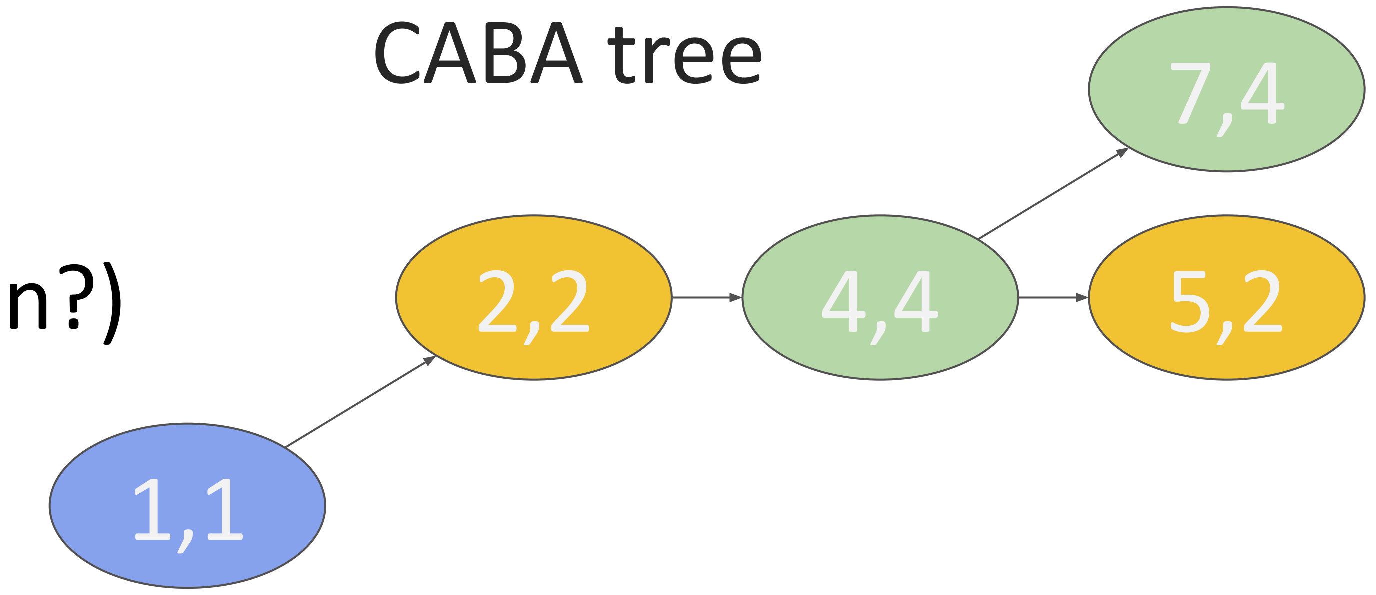
  pros:
  - now we know the only right order of inheritance

  cons:
  - adding CABA in kernel looks unlikely
  - needs “monitor” process (e.g. eBPF)
    - external monitoring is not what CRIU normally does
    - monitor can be killed
    - events can probably be reordered (need extra synchronization?)
  - restore is still quite complex (with multiple helper-processes)

process tree
(pid, sid)

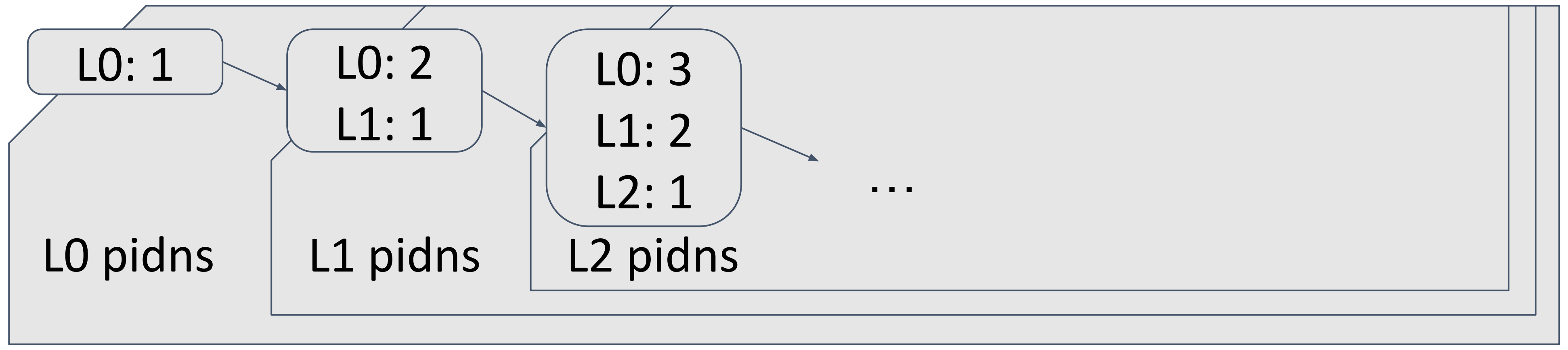


CABA tree

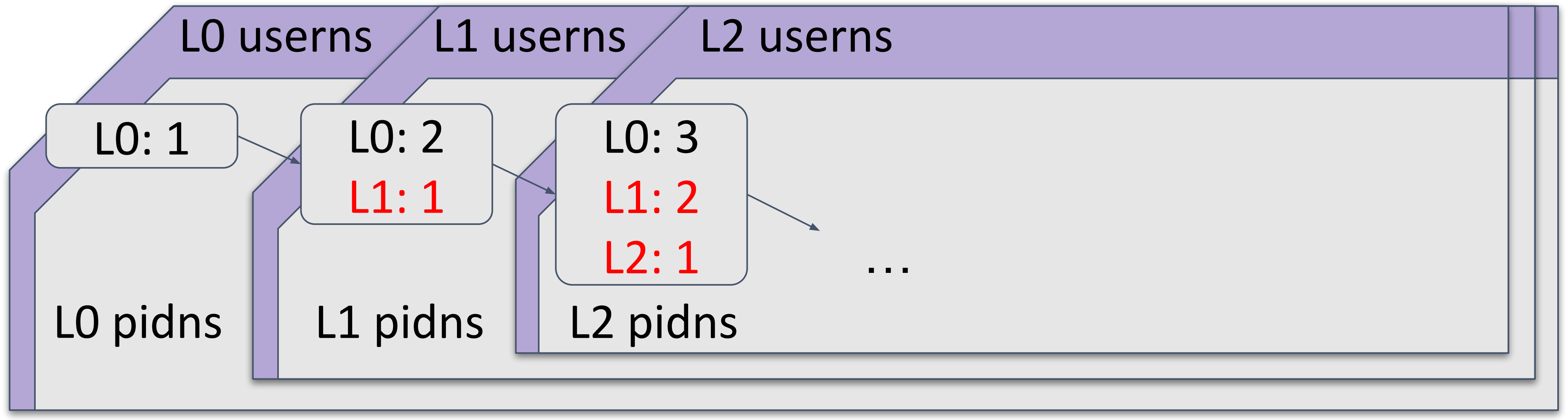


Nested pid and user namespaces restore using clone3 + set_tid syscall

- CRIU should recreate pids of processes on each level of container pidns correctly



- CRIU should recreate pidnses to be owned by specific usernses
- For clone3 CRIU needs checkpoint_restore_ns_capable in owner usernses of each pidns



- CRIU can’t at the same time be in L0 userns and be inside it’s descendant LX userns, to at the same time be capable in all L0..LX usernses for set_tid feature and give LX userns owner to LX pidns

Nested pid and user namespaces restore using clone3 + set_tid syscall

Possible solutions:

looks good: Allow creating a pidns separately from creating its init (unshare + setns + clone3)

- pros:
  - more generic approach
- cons:
  - need to carefully handle cases when two processes try to setns + clone to a not yet fully initialized pidns at the same time
- draft patch (still need testing)

looks bad: provide second (pidns owner) userns in arguments to clone3() syscall

- cons:
  - having two different user namespaces at the same time sounds bad
  - also need to be very careful with using proper userns where needed
- old patch

Ideas, thoughts?

Migration across mismatching CPUs

- Similar thing as in VMs
- CPUID instruction returns the features cpu provides, the task can read it once and then use those cpu features unconditionally
- After migration cpu changes and previously existing features can disappear, using them on other cpu may lead to undefined behaviour e.g.: crash, segfault, memory corruption or out of bound access.
- E.g. in case xsave size is bigger on destination xsave instruction can write out of bounds of provided buffer

Migration across mismatching CPUs

Possible solution:

Similar to ARCH_SET_CPUID we can setup cpuid faulting (X86_FEATURE_CPUID_FAULT) for each container process, and override cpuid instruction result

- how to identify container boundaries? (cpu/cpuid/exec namespace? or seccomp filter?)
- need to carefully handle complex features like X86_FEATURE_XSAVE (e.g. different xsave sizes depend on different other features) and other feature dependencies
- is it enough to override CPUID? what if the process still tries to use cpu feature not shown in CPUID?

Example implementation from OpenVZ
https://github.com/OpenVZ/vzkernel/commit/12d09ff9ef83f38f1908f4be1d6e33f4cd9c8007
https://github.com/OpenVZ/vzkernel/commit/0106a3410c41c6e34c7c4ad8a6d2d146220eaf7c

Ideas, thoughts?

Dumping COW-ed memory effectively

There is no good way to identify that the page is shared with other process via COW

- need to dump page once in CRIU
- can have:
  - multiple processes sharing same COW page
  - reparenting and remap, so that it's not easy to find e.g. child-parent COW

Possible solutions:

1) Add new kcmp() type KCMP_COW
   cons:
     - comparing each two pages of each two processes is ineffective
2) Allow to set per COW page mark from userspace and check it from other copy of this COW page
   cons:
     - breaks on COW page shared between two containers
3) Add flag to identify COW-ed pages in PAGEMAP_SCAN
   pros:
   - fast

   cons:
   - not clear how to find out with what it is COW-ed

Ideas, thoughts?

Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024