State persistence over kexec

Kexec HandOver (KHO)

Why persist state?

You want to update your kernel. But you want to keep

- VMs with VFIO alive
- a dynamic sized in-memory file system
- PCIe PF/VF configuration alive
- driver configuration like flow tables
- memory content of user space applications (CRIU)

Memory persistence proposals

- PRAM (2013)
- PKRAM
- persistent memory pools
- prmem
- KHO + guestmemfs

KHO

- Framework for drivers to hook into
- Serialize/deserialize for state
- Preserves arbitrary memory pages
- Preserves device state
- Similar to Xen breadcrumbs.
  - http://david.woodhou.se/live-update-handover.pdf

KHO building blocks

- KHO FDT passed from old to new kernel
  - Semi-structured driver data
  - Memory ranges to preserve (only non-GFP_MOVABLE)
- Arch specific boot data
  - arm64 appends KHO FDT as “chosen” node
  - x86 adds KHO FDT location to setup_data
- Scratch memory
  - CMA range reserved early at boot by the first kernel
    - only usable for movable allocations to not collide with persisted memory later
  - kexec’ed kernel starts with only scratch memory available
  - After reserving persistent memory, scratch becomes CMA

KHO Device Tree

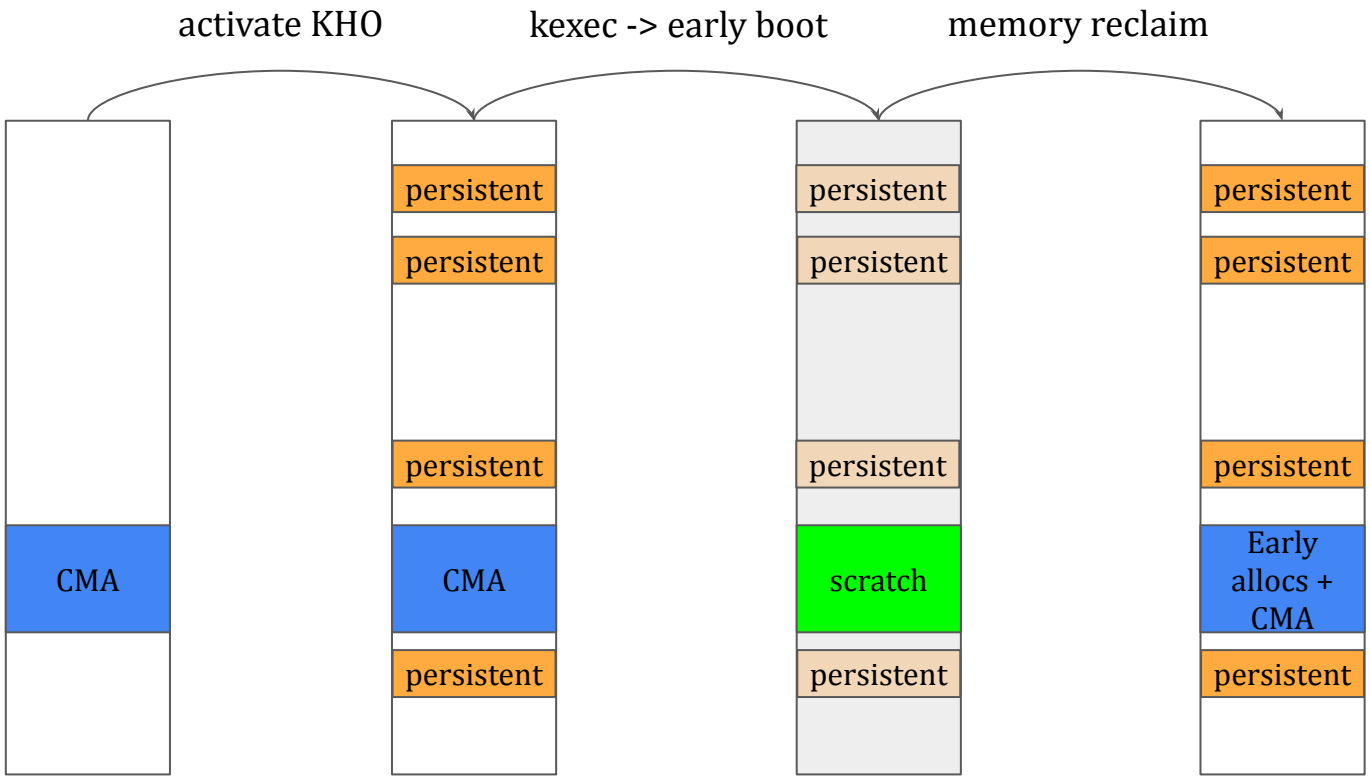
- Flattened Device Tree
- Same file format as system DT, but different content
- Standardized serialization format with
  - Versions for backward compatibility
  - Tooling to describe layout and validate KHO FDT integrity
  - Flexible types

```
ftrace {
  compatible = "ftrace-v1";
  events = < 1 1 2 2 3 3 >;

  global-trace {
    compatible = "ftrace,array-v1";
    trace-flags = < 0x3354601 >;

    cpu0 {
      compatible = "ftrace,cpu-v1";
      cpu = < 0x00 >;
      mem = < 0x101000000 0x38
              0x101000100 0x1000
              0x101000038 0x38
              0x101002000 0x1000 >;
    };
  };
};
```

KHO memory states



Userspace ABI

- `/sys/kernel/kho/scratch_phys`
  - Physical addresses of the scratch areas
- `/sys/kernel/kho/scratch_len`
  - Length of the scratch areas
- `/sys/kernel/kho/active`
  - enable/disable state serialization when `kexec_load_file()` happens

Userspace ABI

- `/sys/kernel/kho/dt_max`
  - Maximal size of the device tree
- `/sys/kernel/kho/dt`
  - The device tree generated during state serialization
- `/sys/firmware/kho/dt`
  - The device tree passed from the previous kernel

Flow - first kernel

- Early boot time
  - Reserve scratch areas
- Late boot time
  - Release scratch areas as CMA blocks to buddy allocator
- User requests serialization
  - `echo 1 > /sys/kernel/kho/active`
  - KHO calls user’s serialization callbacks and creates device tree
  - Data serialized to KHO becomes immutable

Flow - first kernel

- User loads kexec image
  - KHO appends scratch metadata and device tree to kexec image
  - Arch-specific boot information created for KHO
- User requests kexec reboot

Flow - second kernel

- Very early boot
  - setup_arch() parses KHO boot information
  - KHO parses device tree and scratch metadata
  - KHO enables “scratch only” mode for early memory allocations
- Late boot
  - KHO users deserialize their state and claim their memory
  - Scratch memory becomes CMA

Open questions

- Scratch management
  - Initial reservation size
  - Scratch resizing
  - Scratch allocation failure - panic() vs disabling KHO
- Data format
  - FDT all the way
  - Intermediate data structure converted to FDT at kexec_reboot()
  - Completely new data format

Open questions

- State transitions
  - Allow serialization after kexec_load_file()
  - More fine grained states
  - Integration with the driver core
  - Userspace involvment

Thank you