Kexec Handover and Live Update

Pratyush Yadav <pratyush@kernel.org>

Agenda

Agenda

What is Live Update?

High level overview

Building blocks

Upstream status

Future work

Agenda

Agenda

What is Live Update?

High level overview

Building blocks

Upstream status

Future work

What is Live Update?

- It's NOT: live patching, live migration.
- Updates the kernel or hypervisor with minimal disruption for underlying workloads.
- Most commonly used for hypervisors.
- Can also be used by other workloads to reduce kernel patching downtime.
- Multiple cloud providers working together to upstream it.

└What is Live Update?

What is Live Update?
- It's NOT: live patching, live migration.
- Updates the kernel or hypervisor with minimal disruption for underlying workloads.
- Most commonly used for hypervisors.
- Can also be used by other workloads to reduce kernel patching downtime.
- Multiple cloud providers working together to upstream it.

Agenda

Agenda

What is Live Update?

High level overview

Building blocks

Upstream status

Future work

Live update flow

- The system is in normal state.
- The system software starts the live update process.
- Serializes state keeping VMs active but with limited capabilities.
- Pauses VMs and does final serialization.
- Loads and next kernel and hands over the serialized data.
- Next kernel deserializes the data.
- Resumes VM, returning normal operation.

Live update flow

- The system is in normal state.
- The system software starts the live update process.
- Serializes state keeping VMs active but with limited capabilities.
- Pauses VMs and does final serialization.
- Loads and next kernel and hands over the serialized data.
- Next kernel deserializes the data.
- Resumes VM, returning normal operation.

- In “serialization” part, mention the role of system software and kernel.
- Note the similarities to live migration.

What gets preserved?

- VM metadata
- VM memory
- Passthrough devices
- IOMMU mappings

What gets preserved?
- VM metadata
- VM memory
- Passthrough devices
- IOMMU mappings

Agenda

- Complex feature, not easy to do it in one go.
- Upstreaming as a set of building blocks instead.

Agenda
What is Live Update?
High level overview
Building blocks
Upstream status
Future work

Kexec Handover (KHO)

- Creates a mechanism for kernel-to-kernel communication.
- Provides mechanism to mark memory as preserved.
- Makes sure preserved memory does not get used by the next kernel.
- Passes this information over kexec.

Kexec Handover (KHO)
- Creates a mechanism for kernel-to-kernel communication.
- Provides mechanism to mark memory as preserved.
- Makes sure preserved memory does not get used by the next kernel.
- Passes this information over kexec.

- Explain that it is not possible to preserve user memory using this.
- But it can be used for non-liveupdate cases as well, like `reserve_mem` for example.

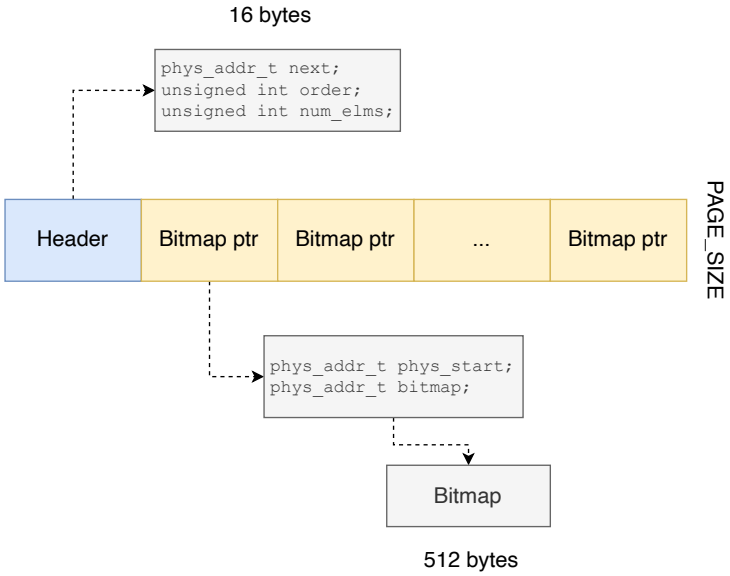
KHO: Memory preservation

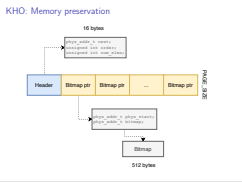
```c
int kho_preserve_folio(struct folio *folio);
int kho_unpreserve_folio(struct folio *folio);
struct folio *kho_restore_folio(phys_addr_t phys);
```


KHO: Memory preservation

int kho_preserve_folio(struct folio *folio);
int kho_unpreserve_folio(struct folio *folio);
struct folio *kho_restore_folio(phys_addr_t phys);

KHO: Memory preservation





KHO: Booting up

- Pre-reserved scratch area for early boot.
- Passing KHO metadata: setup data on x86, chosen node in FDT on arm64.

```
struct kho_data {
        __u64 fdt_addr;
        __u64 fdt_size;
        __u64 scratch_addr;
        __u64 scratch_size;
} __attribute__((packed));
```

```
chosen {
        linux,kho-fdt = <...>;
        linux,kho-scratch = <...>;
};
```


KHO: Booting up
- Pre-reserved scratch area for early boot.
- Passing KHO metadata: setup data on x86, chosen node in FDT on arm64.
struct kho_data { __u64 fdt_addr; __u64 fdt_size; __u64 scratch_addr; __u64 scratch_size; } __attribute__((packed));
chosen { linux,kho-fdt = <...>; linux,kho-scratch = <...>; };

- Mention that kexec image and all early boot allocations go in scratch.
- Mention that chosen node gets set at kexec load time.

KHO: Booting up

- On early boot, only allocate from scratch.

```
enum memblock_flags choose_memblock_flags(void)
{
        if (kho_scratch_only)
                return MEMBLOCK_KHO_SCRATCH;
        [...]
}
```

- After early boot, mark preserved pages as reserved and turn off scratch-only mode
- Reserved pages don't get released to buddy allocator.

KHO: Booting up
- On early boot, only allocate from scratch.
enum memblock_flags choose_memblock_flags(void)
{
if (kho_scratch_only)
return MEMBLOCK_KHO_SCRATCH;
[...]
}
- After early boot, mark preserved pages as reserved and turn off scratch-only mode
- Reserved pages don't get released to buddy allocator.

Live Update Orchestrator (LUO)

- LUO provides a way for userspace to control the live update process.
- Allows marking which resources to preserve.
- Provides a state machine to co-ordinate all the components.
- API is exposed through a set of IOCTLs.

Live Update Orchestrator (LUO)
- LUO provides a way for userspace to control the live update process.
- Allows marking which resources to preserve.
- Provides a state machine to co-ordinate all the components.
- API is exposed through a set of IOCTLs.

- Can't preserve everything since too much state.
- Mention that this is the next layer since it lets userspace actually do stuff.
- Maybe mention that `/dev/liveupdate` can only be opened once and that luod must control it?

LUO: States

```
struct liveupdate_ioctl_set_event {
        __u32           size;
        __u32           event;
};
```

- `LIVEUPDATE_PREPARE`: Normal -> Prepared
- `LIVEUPDATE_FREEZE`: Prepared -> Frozen
- `LIVEUPDATE_FINISH`: Updated -> Normal
- `LIVEUPDATE_CANCEL`: Prepared -> Normal

LUO: States
struct liveupdate_ioctl_set_event {
__u32 size;
__u32 event;
};
- LIVEUPDATE_PREPARE: Normal -> Prepared
- LIVEUPDATE_FREEZE: Prepared -> Frozen
- LIVEUPDATE_FINISH: Updated -> Normal
- LIVEUPDATE_CANCEL: Prepared -> Normal

- Explain all the states.
- FREEZE: Sent from `reboot(2)`.

LUO: File Descriptors

- Userspace can pass in supported file descriptors to LUO to mark them for preservation.
- Not any arbitrary FD, only FDs for supported file types.

```
struct liveupdate_ioctl_fd_preserve {
        __u32                   size;
        __s32                   fd;
        __aligned_u64           token;
};
```


LUO: File Descriptors
- Userspace can pass in supported file descriptors to LUO to mark them for preservation.
- Not any arbitrary FD, only FDs for supported file types.
struct liveupdate_ioctl_fd_preserve {
__u32 size;
__s32 fd;
__aligned_u64 token;
};

- Give some examples of FDs in Linux: memfd, sockets, VFIO, IOMMUFD, KVM, etc.
- Mention some properties that can change with restore FDs, taking memfd as example.
- Mention that the token can be used to identify the FD after reboot.

LUO: Subsystems

- For things that can't be described by a FD.
- Examples: PCI, NVME, ftrace, etc.

LUO: Subsystems
- For things that can't be described by a FD.
- Examples: PCI, NVME, ftrace, etc.

- Mention that not much work done on this so use cases and usage model still unclear.

Memory File Descriptor (memfd)

- memfd attaches a file descriptor to anonymous memory.
- State preserved: memory contents, size and position.
- After preserve, cannot add or remove pages from the memfd.
- Limitations: no sparseness, no swap.

Memory File Descriptor (memfd)
- memfd attaches a file descriptor to anonymous memory.
- State preserved: memory contents, size and position.
- After preserve, cannot add or remove pages from the memfd.
- Limitations: no sparseness, no swap.

- Mention that memfd is the first user of LUO.
- Mention that pages are pinned and holes are filled.

memfd: preservation format

```
/ {
        pos = <0x...>;
        size = <0x...>;
        folios = [array of memfd_luo_preserved_folio]
};
```

```
struct memfd_luo_preserved_folio {
        u64 foliodesc;
        u64 index;
};
```

- Foliodesc: bottom 12 bits for flags, rest for PFN.

memfd: preservation format
/ {
pos = <0x...>;
size = <0x...>;
folios = [array of memfd_luo_preserved_folio]
};
struct memfd_luo_preserved_folio {
u64 foliodesc;
u64 index;
};
- Foliodesc: bottom 12 bits for flags, rest for PFN.

- Explain why we use FDT.

VFIO, PCI, IOMMU, etc. . .

VFIO, PCI, IOMMU, etc. . .

Agenda

Agenda

What is Live Update?

High level overview

Building blocks

Upstream status

Future work

Upstream status

- KHO is in mainline. See `kernel/kexec_handover.c` and `include/linux/kexec_handover.h`.
- LUO v4 sent out few days ago. Patch posting. It is starting to stabilize and is on path to upstream soon.
- memfd support will get merged with the LUO patches.
- RFCs for PCI, VFIO, IOMMU out.

Upstream status
- KHO is in mainline. See kernel/kexec_handover.c and include/linux/kexec_handover.h.
- LUO v4 sent out few days ago. Patch posting. It is starting to stabilize and is on path to upstream soon.
- memfd support will get merged with the LUO patches.
- RFCs for PCI, VFIO, IOMMU out.

Agenda

Agenda

What is Live Update?

High level overview

Building blocks

Upstream status

Future work

Future work

- Supporting more subsystems: huge pages, VFIO, IOMMU, PCI, etc.
- Implementing luod.
- Improving performance for reboots.
- Defining a mechanism for kernels to negotiate versions to enable rollback and roll forward to a wider set of kernels.
- Testing and validation.

Future work

- Supporting more subsystems: huge pages, VFIO, IOMMU, PCI, etc.
- Implementing luod.
- Improving performance for reboots.
- Defining a mechanism for kernels to negotiate versions to enable rollback and roll forward to a wider set of kernels.
- Testing and validation.

Thank you for attending the talk!

Thank you for attending the talk!