memfd preservation using LUO

Pratyush Yadav <pratyush@kernel.org>

Agenda

shmem-backed memfd

HugeTLB-backed memfd

HugeTLB: challenges

Agenda

shmem-backed memfd

HugeTLB-backed memfd

HugeTLB: challenges

shmem-backed memfd

- Merged in mainline now: link
- Pretty simple, dump filemap -> folio mappings in an array.

Agenda

shmem-backed memfd

HugeTLB-backed memfd

HugeTLB: challenges

HugeTLB-backed memfd

- To use huge pages, memfd is backed by hugetlbfs.
- HugeTLB manages its own pages, so some special sauce is needed.
- RFC is out: link.

Agenda

shmem-backed memfd

HugeTLB-backed memfd

HugeTLB: challenges

Problem 1: Hugepage pre-allocation

- On boot, HugeTLB parses cmdline and pre-allocates pages.
- With live update, this might lead to over-allocation.
- Say 100G system, 90 1G hugepages needed. Say 5 1G pages in LUO memfd.
- After KHO boot, HugeTLB allocates 90 pages, then LUO restores the file, adding 5 more.
- System now has 95 hugepages.

Option 1: preserve the whole hstate pool using FLB

Pros:

- Boot time logic simpler. Just skip hstate if in liveupdate.

Cons:

- But more changes needed in hugetlb core to “freeze” hstates after FLB preserve.
- Need to hook into allocation, demotion, compaction, migration paths.

Option 2: count number of hugepages preserved for each hstate

Pros:

- Preservation logic lot simpler.
- Not invasive to core HugeTLB paths.

Cons:

- Need to modify early boot allocation paths.

Problem 2: KHO scratch and hugepage pre-allocation

- Gigantic hugepages (1G) are allocated using memblock.
- On x86, that happens in `setup_arch()` (though we might be able to get rid of that).
- KHO comes up in scratch-only mode. No space in scratch for the huge pages.

Option 1: Disable scratch-only mode earlier

On ARM64 (and every other arch), hugepages are allocated in `mm_core_init()`, right before KHO mem init.

```
void mm_core_init(void) {
        hugetlb_bootmem_alloc();
        // [bunch of code]
        kho_memory_init();
        // disables scratch-only
        memblock_free_all();
```

Do:

```
void mm_core_init(void) {
        // disables scratch-only
        kho_memory_init();
        hugetlb_bootmem_alloc();
```

Option 1: Disable scratch-only mode earlier

- On x86, hugepage allocation happens before `paging_init()`, and KHO needs `page->private`.
- Do (code simplified for presentation):

```
x86_init.paging.pagetable_init();
kho_memory_init();
if (boot_cpu_has(X86_FEATURE_GBPAGES))
        hugetlb_bootmem_alloc();
```

- Or, stop special-casing x86 and remove this from `setup_arch()`.

Option 2: Preserve the entire hugepage pool

- Discussed earlier.
- More complex and invasive in HugeTLB.
- Doesn't solve the problem when HugeTLB files are not preserved.

Problem 3: Early boot access to LUO FLB

- Hugepages are allocated before LUO core is initialized.
- Need FLB data before LUO is up.

Solution: Read-only access to LUO FLB

- Once KHO FDT is known, LUO can parse its data.
- Have early-boot variants that fetch FLB data.
- Directly return serialized data, no retrieve, no memory alloc, etc.
- Caller needs to make sure no side effects.

```
int
liveupdate_flb_incoming_early(struct liveupdate_flb *flb,
                              u64 *datap)
```

Thank you for attending the talk!