mmuse: Memory management of persistent memory in userspace

Giving userspace control over dynamic virtual machine guest memory to survive kexec

James Gowans (jgowans@amazon.com)
David Woodhouse (dwmw2@infradead.org)

Amazon / AWS / EC2

LPC, MM MC, September 2022
Agenda

1. Background, Problem and Requirements
2. Implementation Options
3. Proposal: “mmuse” fs (mem mgmt in userspace)
Live update

Live update of hypervisor via kexec:
serialise -> kexec -> deserialise -> run

Persist guest memory and state across live update (kexec).

Different to snapshot/restore: full restart of userspace process;
new VMM binary! Only preserve guest.

Objective: Make live update properly supported! Starting with memory.
Live update memory management

Memory “use cases:”

- Basic case: fixed allocation at launch.
- Memory overcommit: dynamic allocation/reclaim incl swap
- Deliver faults to userspace for post-copy LM.
- Keep DMA running during kexec: IOMMU persistent pgtables
- Pass through slice of PCI BAR
- Side-car VM: carve out portion of memory to run another VM.
Options considered

**Fully kernel management persistence:** Traditional kernel-driven allocations. Kernel state passed from old to new kernel. Like pkram\(^1\) RFC; think `tmpfs` with persistence.
State can be passed similar to Xen breadcrumbs.
Pros: fast, Cons: complex state hand over.

**Filesystem with userspace control:** Filesystem backed by non-kernel managed memory: hard split of persistent vs ephemeral.
Provide privileged userspace process the ability to control memory mappings (`pg_offset` to `PFN`) of files.
Store arbitrary file types: memory, state, pgtables, etc.

\(^1\)https://lwn.net/Articles/851192/
Suggestion: filesystem with userspace control

Idea floated at LSF/MM earlier this year:
https://lwn.net/Articles/895453/

Suggesting userspace filesystem. Justification:

- Avoid complexity of passing and re-hydrating state.
- Avoid attempting to expose allocation policies and things like swap to persistent memory.
- Allow userspace policy and implementation to develop freely.
Proposal: “mmuse” fs (mem mgmt in userspace)

- Carve out persistent memory by \texttt{mem=} cmdline param:
  
  host physical addr space:

  \begin{tabular}{|c|c|}
    \hline
    kernel & managed \text{Guest memory} \\
    \hline
  \end{tabular}

- Mount \texttt{mmuse}, setting backing file to something with access to the carved out memory: /dev/mem, DAX device, etc.

- Control process: own persistent memory, create files, program allocations into kernel via file ioctls.

- Client process (QEMU): use those allocation in non-privileged way: just open the file.
Example: Set up filesystem and backing memory. Mount:

```bash
mount -t mmuse guest-memory /mnt/guest-memory
```

Initially admin file for control process:

```bash
ls -l /mnt/guest-memory/
-rw-rw---- 1 root root 0 Aug 4 00:00 admin
```

Set backing to `/dev/mem`:

```c
int admin_fd = open("/mnt/guest-memory/admin")
int devmem_fd = open("/dev/mem")
ioctl(admin_fd, SET_BACKING_FD, devmem_fd)
```
Example: Programming a mapping from backing memory to a file:

```c
dst_fd = open("/mnt/guest-memory/dom123_0_3GiB");
struct mmuse_mapping mapping = {
    .dst = dst_fd,
    .src_start = 100 * GiB,
    .size = 3 * GiB,
    .dst_start = 0,
    .granularity = GIGANTIC_PAGE // lvl 3 -> 1 GiB
};
ioctl(admin_fd, MAP_MEMORY_RANGE, &dst_fd);
```

When client processes mmaps that file it would get backing memory, faulting in 1 GiB PTEs.
Control process would replay after live update.
To persist or not to persist?

Should the filesystem preserve state internally across kexec: files, mappings, etc. Or should userspace re-drive filesystem state? Userspace need to know state anyway, so ought to be able to re-drive.

Advantage of persisting: 1) faster restore on LU, 2) kernel can consume files early.

Advantage of not persisting: No need for memory for pmem for metadata. Generally simpler.

Suggest: no persist at first, retrofit when more stable.
Discussion and Questions

Work so far:
Fairly complete proof of concept implemented. Not LKML worthy yet, but could be!

Open floor for questions/comments.

Some ideas for feedback:
- Are we re-inventing or overcomplicating this?
- Other use-cases than live-update?
- Other ideas to solve this which we should look at?
- Should we add mmuse to linux?
Backup slides
Example: hierarchy using one mmuse file as backing memory for another:

```c
int source_fd = open("/mnt/guest_memory/dom:123_memory")

mount -t mmuse dom:123_memory /mnt/dom:123_memory
int admin_fd = open("/mnt/dom:123_memory/admin")

ioctl(admin_fd, SET_BACKING_FD, source_fd)
```

Use case: hand over large chunk of memory to guest VMM. That VMM can carve it up for sidecar VMs.
Example: memory overcommit

Memory overcommit: reclaim a chunk of memory currently assigned to a file:

```c
struct mappings = {
    .dst = dst_fd,
    .src_start = (100 << 30),
    .dst_start = 0,
    .size = (16 << 20)
}
ioctl(admin_fd, UNMAP_MEMORY_RANGE, &dst_fd);
```

Use case: hand over large chunk of memory to guest VMM. That VMM can carve it up for sidecar VMs.