

~~PCI core Live Update Roadmap~~

Preserving tmpfs Files Across Live Update

David Matlack <dmatlack@google.com>

The Problem

Goal: Preserve a file on a tmpfs mount across a Live Update (kexec), so that it is still reachable by its path afterwards.

- Motivating use case: VMM binaries, logs, and other files kept in host memory.
 - Hosts have no local storage.
 - Binaries are fetched from network into memory and then must be preserved across kexec.
 - Static persistent filesystems (e.g., pmem+fsdax+ext4) are too inflexible
- LUO can already preserve a **memfd** today.
- But a memfd has no name or path. Anything that needs a filesystem path cannot use it.

Preserving Is Easy

memfd_luo already knows how to preserve the contents of any shmem inode. The only thing stopping it from accepting a tmpfs file is a check that the file is unlinked:

```
static bool memfd_luo_can_preserve(struct liveupdate_file_handler *handler,
                                   struct file *file)
{
-   struct inode *inode = file_inode(file);
-
-   return shmem_file(file) && !inode->i_nlink;
+   return shmem_file(file) || shmem_mapping(file->f_mapping);
}
```

- From: "[mm/memfd_luo: Expand support beyond memfd to all generic shmem inodes](#)" (RFC).
- File contents, size, and position are preserved with the existing memfd ABI. **No new KHO ABI needed.**

Retrieving Is Hard

LUO retrieve today means: **the kernel creates a new file and returns an fd.**

```
static int memfd_luo_retrieve(struct liveupdate_file_op_args *args)
{
    ...
    file = memfd_alloc_file("", MFD_ALLOW_SEALING);
    ...
    memfd_luo_retrieve_folios(file, folios_ser, ser->nr_folios);
    args->file = file;
}
```

- The result is always an **anonymous inode** (unlinked memfd).
- There is no way to say where the file should live:
 - Which tmpfs mount, and with what mount options?
 - What path (name, parent directories)?
 - What ownership, permissions, xattrs, etc.?

Who reconstructs the filesystem around the preserved folios, and how do the folios get into it?

Three Proposals

1. **RETRIEVE_INT0** — Generic extension to LUO. Userspace provides the file into which LUO retrieves the preserved contents.
→ [\[RFC PATCH 0/5\] liveupdate: LIVEUPDATE_SESSION_RETRIEVE_INT0_FD](#) (*David Matlack*)
2. **tmpfs mount/file preservation** — The kernel preserves the mount and files in root directory as first-class LUO objects and recreates them. Userspace flattens and un-flattens mount for preservation/retrieval.
→ [\[RFC PATCH 0/6\] lu0: tmpfs preservation](#) (*Pratyush Yadav*)
3. **Move folios from memfd to tmpfs** — Retrieve an anonymous memfd as today, then zero-copy move its contents into a tmpfs file with splice(SPLICE_F_MOVE).
→ *No RFC on the mailing list yet (Pasha Tatashin)*

Running Example

A file in a subdirectory, on a tmpfs with non-default mount options:

```
mount -t tmpfs -o size=4G,huge=always,noswap tmpfs /tmpfs  
/tmpfs/vm1/bin/vmm
```

Proposal 1: RETRIEVE_INT0

The kernel injects the preserved folios into an **empty** shmeme file provided by userspace.

- RFC adds a new ioctl: LIVEUPDATE_SESSION_ **RETRIEVE_INT0_FD**.
- Could instead be a flag on the existing LIVEUPDATE_SESSION_ **RETRIEVE_FD**

```
system("mount -t tmpfs -o size=4G,huge=always,noswap tmpfs /tmpfs");

fd = open("/tmpfs/vm1/bin/vmm");
preserve(session, fd, TOKEN_FILE);           /* any path is fine */

kexec();

system("mount -t tmpfs -o size=4G,huge=always,noswap tmpfs /tmpfs");

fd = open("/tmpfs/vm1/bin/vmm", O_CREAT);
retrieve_into(session, TOKEN_FILE, fd);       /* kernel fills contents */
```

Proposal 1: RETRIEVE_INT0 — Trade-offs

Pros

- LUO only preserves what userspace cannot recreate: file contents (memory)
- No new KHO ABI
- No new VFS changes
- Any filesystem topology, any mount options, any user namespace; free via existing syscalls
- Zero-copy: folios go straight from KHO into the target file
- Generic LUO primitive, reusable by any file handler. Potential future use-cases:
 - Mount preservation (where to retrieve the mount)
 - HugeTLBfs file preservation (where to retrieve the file)
 - VFIO cdev preservation (eliminate special-case [vfio_device_liveupdate_cdev_open\(\)](#))

Cons

- New LUO uAPI (new ioctl, or extension to existing ioctl)
- Requires validating target file before filling contents (must be empty shmem file)

Proposal 2: tmpfs Mount/File Handlers

Allow tmpfs mounts and files (in root directory) to be preserved as separate LUO objects. On retrieve, the kernel creates an **anonymous (detached) tmpfs mount** and the **files in its root**.

```
system("mount -t tmpfs tmpfs /tmpfs");
mnt = open("/tmpfs", O_DIRECTORY);
preserve(session, mnt, TOKEN_MNT);           /* mount first */

fd = open("/tmpfs/vmm");
preserve(session, fd, TOKEN_FILE);           /* name ("vmm") + mode saved */

kexec();

mnt = retrieve(session, TOKEN_MNT);           /* detached tmpfs */
move_mount(mnt, "/tmpfs");                  /* mount at /tmpfs */

fd = retrieve(session, TOKEN_FILE);           /* creates /tmpfs/vmm */
```

- Preserved: mount block limit; file name, size, pos
- **Limited to regular files in the mount root** (no subdirectories)
- **Mount options not preserved**

Proposal 2: What about Subdirectories and Mount Options?

To preserve files in subdirectories: Userspace flattens into the root before preserving, then unflatten after retrieving.

To restore mount options: Userspace remounts after retrieve.

```
system("mount -t tmpfs -o size=4G,huge=always,noswap tmpfs /tmpfs");

mnt = open("/tmpfs", O_DIRECTORY);
preserve(session, mnt, TOKEN_MNT); /* saves size=4G only */

link("/tmpfs/vm1/bin/vmm", "/tmpfs/vm1_bin_vmm"); /* flatten */
fd = open("/tmpfs/vm1_bin_vmm");
preserve(session, fd, TOKEN_FILE); /* saves "vm1_bin_vmm" as name */

kexec();

mnt = retrieve(session, TOKEN_MNT);
fd = retrieve(session, TOKEN_FILE); /* creates <detached-mount>/vm1_bin_vmm */

remount(mnt, {"huge=always", "noswap"});
renameat(mnt, "vm1_bin_vmm", mnt, "vm1/bin/vmm"); /* unflatten */
move_mount(mnt, "/tmpfs");
```

Proposal 2: tmpfs Handlers — Trade-offs

Pros

- Simplest for userspace for a flat tmpfs with default options
- Zero-copy

Cons

- New LUO uAPI commitment (tmpfs mount becomes a supported LUO file type)
- New KHO ABI objects (tmpfs mount, tmpfs file)
- New KHO ABI is redundant (userspace still needs an unflattened manifest)
- New VFS changes (kernel-created anonymous/detached mounts)
- A user in a bind mount cannot preserve or retrieve its own files (no access to mount root)
- Some mount options can't be set by remount today (noswap, quotas)
- Remount and unflattening after kexec is extra work that lands in the blackout window

Proposal 3: Retrieve as memfd, Then Move to tmpfs

Preserve the tmpfs file with the memfd handler. Retrieve it as an anonymous memfd, then **move** its folios (zero-copy) into a userspace-created tmpfs file with splice(SPLICE_F_MOVE).

```
system("mount -t tmpfs -o size=4G,huge=always,noswap tmpfs /tmpfs");
fd = open("/tmpfs/vm1/bin/vmm");
preserve(session, fd, TOKEN_FILE);

kexec();

system("mount -t tmpfs -o size=4G,huge=always,noswap tmpfs /tmpfs");
src_memfd = retrieve(session, TOKEN_FILE);
dst_tmpfs = open("/tmpfs/vm1/bin/vmm", O_CREAT);

pipe(pipe_fds);
while (off < size) {
    n = splice(src_memfd, &off, pipe_fds[1], NULL, pipe_size, SPLICE_F_MOVE); /* memfd -> pipe */
    splice(pipe_fds[0], NULL, dst_tmpfs, &doff, n, SPLICE_F_MOVE); /* pipe -> tmpfs */
}
close(src);
```

Proposal 3: What It Would Take

Read side: new shmem-specific **try_steal()** to transfer folios from file to pipe

- *Fail if the folio is mapped, pinned (GUP/DMA), or swapped*
- *Refuse if sealed (F_SEAL_WRITE, F_SEAL_FUTURE_WRITE, F_SEAL_SHRINK)*
- *Remove folios from shmem file, release accounting, leave hole in file*

Write side: new shmem-specific **splice_write()** that transfers stolen folios from pipe to file

- *shmem_add_to_page_cache(), charge destination limits and quotas*
- *Fall back to copying per buffer when stealing fails*

Large folios: Moving large folios through a pipe would need to be solved

- *Default max pipe size for unprivileged users is 1MB*
- *splice_folio_into_pipe() splits folios into PAGE_SIZE pipe buffers*

New destructive contract: SPLICE_F_MOVE is not destructive today.

- *It is a hint to only dropped clean page cache copies of disk blocks.*

Proposal 3: Trade-offs

Pros

- General-purpose shmem → shmem move, potentially useful beyond LUO
- No changes to LUO or KHO ABI

Cons

- New uAPI (destructive splice())
- Large change in core code: splice, pipe buffers, shmem page cache and accounting. More maintainers involved.
- Large folios are split, or pipes need a redesign.
- Must handle live-file hard cases: mapped, pinned, swapped, concurrent access, transfer folios and accounting across mounts.
- Double work in the blackout window: folios inserted into the memfd, then moved again to tmpfs file.

Comparison

	RETRIEVE_INT0	tmpfs handlers	memfd + move
New uAPI	New LUO ioctl or flag	New LUO file type (tmpfs mount)	Destructive splice()
New KHO ABI	None	tmpfs mount + tmpfs file	None
New VFS changes	None	Detached mounts	Move op on shmem
Filesystem topology owner	Userspace	Duplicated (Kernel + Userspace)	Userspace
Subdirs / mount options	Free	Mostly (noswap/quotas needs work)	Free
Preserves file-mount	No, but can be added later	Yes	No
Extra work in blackout	No	Yes (remount, unflatten)	Yes (move)
Zero-copy	Yes	Yes	Yes
Works on live files	Not needed	Not needed	Required (hard)
Keeps large folios	Yes	Yes	No (or pipe redesign)

Proposal & Discussion

Proposal: Use RETRIEVE_INT0 as the core primitive for retrieving non-anonymous files, starting with tmpfs.

Preserve tmpfs mounts if-and-when it's needed using RETRIEVE_INT0.

- Is the simple case (flat tmpfs, default options) enough to justify new KHO ABI and VFS surface for the tmpfs handlers?
- Is a general zero-copy move for shmem worth its complexity, when RETRIEVE_INT0 solves the LUO case without the hard parts?
- Should RETRIEVE_INT0 be a new ioctl or a flag on the existing retrieve ioctl?