FUSE mount recovery

Stéphane Graber
Owner, Zabbly
stgraber@stgraber.org

Aleksandr Mikhalitsyn
Kernel engineer, Canonical
aleksandr.mikhalitsyn@canonical.com

Motivation

1. Fuse daemon crash may happen
    a. In LXC project we heavily rely on LXCFS filesystem that “virtualizes” some procfs/sysfs files like loadavg, cpuinfo, etc
    b. Cephfs-fuse, glusterfs, etc
2. Container checkpoint/restore with fuse daemon inside [2]

Our first approach to the problem

This year I have tried to propose & implement a PoC of API that enables fuse mounts healing [3], [4]

- [RFC PATCH 0/9] fuse: API for Checkpoint/Restore https://lore.kernel.org/all/20230220193754.470330-1-aleksandr.mikhalitsyn@canonical.com/#r
- I have received no objections against the idea but implementation was criticised because of one hacky thing which is a key problem that I want to cover here
- Thanks to Bernd Schubert, Miklos Szeredi and Christian Brauner for review and comments on that

Problem of crashed fuse daemon

- Each active fuse mount superblock has an associated fuse connection
- When fuse daemon crashes, daemon process exits, fuse connection file descriptor gets closed and from that point fuse superblock is not recoverable.

First idea

Okay, let’s just hold this fuse connection file descriptor somewhere in addition to a fuse daemon (parent process, unix socket) to keep a ->f_count >= 1 all the time

First idea (cont.)

And yes it works, but:

- When a new daemon will be started it will expect to receive a FUSE_INIT request from the kernel
  - … and reply with important information about fuse protocol version, implemented features, etc
- We need some way to tell the kernel that we need to start everything from scratch but keep mounts

Second idea: ioctl(FUSE_DEV_IOC_REINIT)

- Let’s introduce a ioctl(FUSE_DEV_IOC_REINIT), this handle will carefully cleanup in-kernel state of fuse connection that related to a daemon and user space state, but keep everything else in place
  - Runtime fuse connection flags like: no_open, no_flock, etc. All of these can be safely cleaned as it will be properly reinitialized later on
  - Drop all non processed fuse requests that can be sent to a previous fuse daemon process
  - Send FUSE_INIT request to a new user space daemon

Any problems left?

- Yes, because if you try this approach with any daemon that based on libfuse your new daemon will crash almost immediately.
- Why? Because eventually it will receive a fuse request from the kernel about some inode that was used by a previous daemon (and ino was allocated for it in the user space), but new daemon does not know anything about it. Libfuse contains appropriate checks for that and that’s good.
- Another problem is file descriptors that were opened before a new daemon was started

=> we need some way to invalidate all “old” stuff and replace it with a new one

Dentry revalidation

- There is a callback ->d_revalidate(dentry, flags) the purpose of it is to tell VFS lookup machinery if these dentry is a still valid to use or not.
- Fuse filesystem implement it as fuse_dentry_revalidate and idea was to extend it and add a concept of “generation” for a fuse connection. When FUSE_DEV_IOC_REINIT is called then connection generation increases, but each fuse_inode keeps its own value for connection generation (filled in the fuse_alloc_inode callback)
- If these values are not the same at the point when fuse_dentry_revalidate is called then this dentry reported as stale dentry to VFS, VFS will release it and go to a slow lookup path that will force a new dentry allocation and then a new inode allocation/lookup for it

Unfortunately, that’s not all :)

- Dentry revalidation solves a problem of a new daemon crashing perfectly well, because before a fuse request about a stale inode will go to the userspace stale dentry will be invalidated and fuse request won’t go. Instead the userspace will receive a FUSE_LOOKUP request from the kernel and everything will be handled correctly.
- But… if you have a bind mounts of your fuse mount you are in trouble.
  - Struct vfsmount has ->mnt_root field that points to a root dentry of a mount. For a first mount we have ->mnt_root = sb->s_root. For non-root bindmounts ->mnt_root points to some dentry inside the filesystem tree.
  - Obviously, these dentries will become invalid after FUSE_DEV_IOC_REINIT call and… our bindmounts will be broken

Next idea: ioctl(FUSE_DEV_IOC_BM_REVAL)

- Take a list of mounts for a superblock sb->s_mounts
- Iterate over them, calculate dentry_path for each ->mnt_root
- Use it with vfs_path_lookup to find a new (and valid) dentry for the same path
- Update ->mnt_root with a new dentry

And… this is too hacky. And this is a main problem for now.

What’s next?

- Use MOVE_MOUNT_BENEATH instead of bind mount revalidation?
- => if it works then make integration with libfuse and publish a new version of kernel series.

Questions ?

Stéphane Graber
Owner, Zabbly
stgraber@stgraber.org

Aleksandr Mikhalitsyn
Kernel engineer, Canonical
aleksandr.mikhalitsyn@canonical.com

Links

[1] Linux kernel patchset https://github.com/mihalicyn/linux/commits/fuse_cr_rev2

[2] Bringing up FUSE mounts C/R support https://lpc.events/event/16/contributions/1243/

[3] [RFC PATCH 0/9] fuse: API for Checkpoint/Restore
https://lore.kernel.org/all/20230220193754.470330-1-aleksandr.mikhalitsyn@canonical.com/#r

[4] [RFC PATCH v2 0/9] fuse: API for Checkpoint/Restore
https://lore.kernel.org/all/20230403144517.347517-1-aleksandr.mikhalitsyn@canonical.com/