Files on Unmounted Mounts

Bhavik Sachdev
Pavel Tikhomirov

東京 2025
LINUX PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Problem

For any given file descriptor, CRIU needs to know from which mount it was opened from, so that it can open the fd from the same mount during restore.

We can get the `mnt_id` for this mount from `/proc/<pid>/fd/<fdinfo>` and information about the mount from `/proc/<pid>/mountinfo`.

However, in case the fd is on an "unmounted" mount (`umount2(mnt, MNT_DETACH)`), mount information is no longer present in `/proc/<pid>/mountinfo`. `statmount()` also does not work on these mounts, since they do not belong to any mount namespace.

So, we need some way to get mount info in order to support C/R for files on "unmounted" mounts.

Unmount Namespace Approach and Why it Failed

- Create a special kernel-only “unmount namespace” where all “unmounted” mounts would be moved to.
- Modify `statx()` to also export `mnt_ns_id`, so that we can do the following:

```
statx(fd, “”, AT_EMPTY_PATH, STATX_MNT_ID_UNIQUE |
STATX_MNT_NS_ID, &stat);
struct mnt_id_req req { // also specify .size and .param
    .mnt_id = stat.stx_mnt_id;
    .mnt_ns_id = stat.stx_mnt_ns_id;
};
statmount(&req, &statmount_buf, buf_size, 0);
```

Unmount Namespace Approach and Why it Failed

We sent out a patchset that implemented this approach:
https://lore.kernel.org/all/20251002125422.203598-1-b.sachdev1904@gmail.com/
However, it had lots of problems:

- Our implementation of cleaning up mounts from umount_mnt_ns was incorrect and modified performance sensitive code.
- It's weird to be able to `statmount()` on a mnt_id of a unmounted mount that `listmount()` won't surface.
- Complicated locking in `umount_tree()` and `mntput_no_expire()` (because we had to manage mounts in `umount_mnt_ns`), which was unacceptable.

Christian pointed all of these out here and more:
https://lore.kernel.org/all/20251006-erlesen-anlagen-9af59899a969@brauner/

fd-based statmount()

Introduce a new `STATMOUNT_BY_FD` flag and modified struct mnt_id_req like this:

```
struct mnt_id_req {
    __u32 size;
    union {
        __u32 mnt_ns_fd;// for listmount
        __u32 mnt_fd;
    }
    __u64 mnt_id;
    __u64 param;
    __u64 mnt_ns_id;
};
```

statmount would now accept a file descriptor and return mountinfo for the mount this fd was on.

This works even if the mount is unmounted. We can get all information except mount point and mount namespace id, so we unset flags for those mounts in the returned struct i.e `STATMOUNT_MNT_POINT` and `STATMOUNT_MNT_NS_ID` are unset.

fd-based statmount()

- It was simpler to implement than `umount_mnt_ns` approach.
- Makes it easier to get mountinfo for any mount via fd, even if it is not on a unmounted mount.
  - Instead of getting mnt_id_unique from `statx()` and `mnt_ns_id` from `ioctl()` (`NS_MNT_GET_NEXT` and `NS_MNT_GET_PREV`) we can directly get mountinfo using `STATMOUNT_BY_FD`.
- We sent out 6 versions of the fd-based approach.
- This (v7) approach has been accepted and will be in linux-next soon.

https://lore.kernel.org/all/20251204-mondfinsternis-sowohl-fd9f566ae1d0@brauner/

Permission Checks for fd-based statmount()

- `statmount()` requires the user to be `CAP_SYS_ADMIN` in the user namespace to which the mount namespace belongs to.
- In case of file descriptors we decided upon not having permission checks because of the following reasons:
  - All filesystem related information is already available via `fstatfs()` without any permission checks
  - Mount information is also available via `/proc/pid/mountinfo` (without any capability check)
  - Given that we have access to a fd on the mount which means we had access to the mount at some point (or someone that had access gave us the fd). So, we should be able to access mount info.

C/R of files on unmounted bind mounts: Dump Stage

Currently, we have only added support for unmounted mounts that are a bind mount of a regular mount, using (`fd-based statmount()`):
https://github.com/checkpoint-restore/criu/pull/2754 (WIP)

- CRIU constructs a list of mounts called `mntinfo` before it starts dumping fds.
- So, if during fd dump, we get a `mnt_id` not in the `mntinfo` list, we assume it to be a unmounted mount.
- We then try to get mount info using `fd-based statmount()` and mark this mount to be a unmounted mount in the image.

C/R of files on unmounted bind mounts: Restore Stage

Restoration of unmounted bind mounts takes place outside the general mount restore process.

- We create a list of `detached_mounts` during collect phase.
- For each mount, we create a temporary mount point (since in case, of unmounted mounts we can't know mount points). `/.criu.detached.XXXXXX`
- Mount filesystems at these mount points using the original mount.
- While opening fds, we open them at `/.criu.detached.XXXXXX/<filename>`.
- After all fds have been opened we umount using `MNT_DETACH` and remove our temporary mount points.

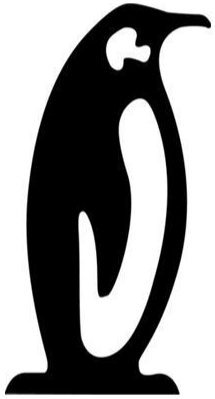
NOTE: "detached" mounts and "unmounted" mounts mean the same thing (in this context), we will update the CRIU Pull Request and replace "detached" -> "unmounted".

Future Work

- A way to export mountinfo for anonymous mounts through the kernel.
  - Anonymous mounts can be created using the `fsmount()` syscall or with `open_tree()` called with `OPEN_TREE_CLONE`.
- Checkpoint/Restore of filesystem of unmounted mount.
  - Currently, we only C/R if the unmounted mount is a bind mount of a regular mount (we use the filesystem of the regular mount).
  - We won't have access to root of the filesystem of the unmounted mount, so we would have to do some sort of partial dumping.
    - Recreate only those files and subdirectories we have access to.

Thank You

Questions?



東京 2025
LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025