Files on Detached Mounts

Mentors: **Pavel Tikhomirov**

May 11, 2025

Contents

1 Abstract

Whenever we attach a disk to our computer, we need to know a couple of things before we can access the data from it.

1. **File System**
   Governs file organization and access (`ext2, ext4` are some examples)
2. **"Location"**
   All devices attached to the computer appear as "special" files inside the `/dev` directory. For example, a partition of my SSD is mounted at `/dev/nvme0n1p1`

Say we want to access some data from one of these devices; we will first have to attach them to the **"Root File System"**. To do this, we use the `mount` syscall. After we finish using the disk (device), we can remove it from the **"Root File System"**. For that, we use the `umount` syscall. The `umount` syscall provides a flag called `MNT_DETACH`. `MNT_DETACH` makes the device unavailable for new access; however any opened file(s) are still valid. Unmounting in such a way is called lazily unmounting the drive. CRIU currently does not support checkpoint/restore of such applications.

2 Technical Details

2.1 Problem with Lazily Unmounted `fds`

Currently if you try to checkpoint the program given below:

```
int main(void)
{
    char *source = "/home/bsach/tmp";
    char *dst = "/home/bsach/Code/transport";
    mount(source, dst, NULL, MS_BIND, NULL);
    int fd = open("/home/bsach/dumb/hello", O_CREAT | O_WRONLY);
    char *data = "hello\n";
    int count = write(fd, data, strlen(data) + 1);
    umount2(dst, MNT_DETACH);
    printf("have lazily umounted\n");
    printf("waiting for C/R\n");
    while(1);
}
```

(complete program here), you get the following error:

```
(00.018117) Error (criu/files-reg.c:1790): Can't lookup mount=2943 for fd=3 path=/hello
(00.018164) Error (criu/cr-dump.c:1681): Dump files (pid: 99810) failed with -1
```

This happens because CRIU gets `mnt_id` (unique id of a mount) from `/proc/$pid/fdinfo/$fd`. It then looks for this `mnt_id` in `/proc/$pid/mountinfo/`. But, when a mount gets unmounted (lazily or otherwise), its information (major, minor dev number, etc.) is no longer present in `/proc/$pid/mountinfo`. All these details are necessary to recreate the file during restoration.

2.2 New Kernel Syscall to the Rescue: statmount

`statmount` was introduced in the 6.8 kernel. `statmount` returns to us the details for a given `mnt_id`. Here's the output if you call `statmount` on a random `mnt_id`:

```
$ sudo ./listing
mnt_id:          2147483677
mnt_parent_id:   2147483681
fs_type:         proc
mnt_root:        /
mnt_point:       /proc
mnt_opts:        proc
sb_dev_major:    0
sb_dev_minor:    25
```

We end up getting all the info available at `/proc/$pid/mountinfo` in the form of a nice `struct`, which is way better (and faster) than parsing a string. The best thing, however, is that we can also use these syscalls to find information about unmounted (lazily or otherwise) mount points (atleast for kernels newer than 6.7).

2.3 Finding mountinfo for a file on detached mount

Given a fd, we can get call `statx` on it to find the mount ID of the mount containing this file. Specifically, `stx_mnt_id` with the `STATX_MNT_ID_UNIQUE` flag gives us this info. Now, we can pass this mount ID to `statmount`. Here's a small program that does that:

```
int main(void)
{
    #define STATMOUNT_BUFFER_SIZE 4096
    char *source = "/home/bsach/tmp";
    char *dst = "/home/bsach/Code/transport";
    mount(source, dst, NULL, MS_BIND, NULL);
    int fd = open("/home/bsach/dumb/hello", O_CREAT | O_WRONLY);
```

```
    char *data = "hello\n";
    int count = write(fd, data, strlen(data) + 1);
    umount2(dst, MNT_DETACH);
    printf("have lazily umounted\n");
    struct statx stat;
    statx(fd, NULL, AT_EMPTY_PATH, STATX_MNT_ID_UNIQUE, &stat);
    struct mnt_id_req req = {
        .size = sizeof(req),
        .mnt_id = stat.stx_mnt_id,
        .param = STATMOUNT_SB_BASIC | STATMOUNT_MNT_BASIC
        | STATMOUNT_PROPAGATE_FROM
                | STATMOUNT_MNT_ROOT | STATMOUNT_MNT_POINT
        | STATMOUNT_FS_TYPE,
    };
    struct statmount *stmnt = malloc(STATMOUNT_BUFFER_SIZE);
    statmount(&req, stmnt, STATMOUNT_BUFFER_SIZE, 0);
    printf("mnt_id:\t\t%" PRIu64 "\nmnt_parent_id:\t%" PRIu64 "\n"
           "fs_type:\t%s\nmnt_root:\t%s\nmnt_point:\t%s\nmnt_opts:\t%s\n",
           (uint64_t)stmnt->mnt_id,
           (uint64_t)stmnt->mnt_parent_id,
           stmnt->str + stmnt->fs_type,
           stmnt->str + stmnt->mnt_root,
           stmnt->str + stmnt->mnt_point,
           stmnt->str + stmnt->mnt_opts);

    printf("sb_dev_major:\t\t%" PRIu64 "\nsb_dev_minor:\t\t%" PRIu64 "\n",
            (uint64_t)stmnt->sb_dev_major,
            (uint64_t)stmnt->sb_dev_minor
    );
}
```

```
$ sudo ./simple
have lazily umounted
mnt_id:         2147483681
mnt_parent_id:  2147483650
fs_type:        ext4
mnt_root:       /
mnt_point:      /
mnt_opts:       ext4
sb_dev_major:   259
sb_dev_minor:   6
```

We would like to do something similar if, for a fd, we cannot find its details in `/proc/$pid/mountinfo`, we instead use `statx` and `statmount`. **The assumption here is that we have an fd at a `mnt_id` but it's details are not present in `/proc/$pid/mountinfo` then the fd belongs to a detached mount.**

3 Implementation

3.1 The General Plan

3.1.1 Checkpointing

1. When we can't find info about a `mnt_id` in `/proc/$pid/mountinfo` we obtain its details using `statmount`.
2. Identify that the `mnt_id` belongs to a detached mount.
3. Mark the fd to belong to detached mount in the image and store all info obtained from `statmount`.

3.1.2 Restoring

1. Create a bind mount using the info in the image.
2. Open the file in the same way as CRIU would open any other file.

3. Lazily unmount the mount point.

3.2 Identifying a detached mount

CRIU stores all mount-related information in a `struct mount_info`. CRIU stores all mounts-related info in a list called `mntinfo` (a global variable). Functions like `dump_one_reg_file`, call `lookup_mnt_id` to get mount details (from the list) in the form of a `struct mount_info`.
For a simple solution (for systems running a kernel with support for `statmount`), we can do the following:

```
int dump_one_reg_file(int lfd, u32 id, const struct fd_params *p)
{
    /* ... */

    mi = lookup_mnt_id(p->mnt_id);
    if (mi == NULL) {
        if (/* .... */) {
            /* .... */
        } else if (is_in_detach_mount(lfd)) {
            /* pass */
        } else {
            /* ... */
        }
    }

    /* ... */
}
```

Here's what the `is_in_detach_mount` function should do:

1. Call `statx` on the `fd` with `STATX_MNT_ID_UNIQUE`.
2. Call `statmount` with `statx_mnt_id`.
3. Dump the info we get from `statmount`.
4. Add a new `struct mount_info` to `mntinfo`, so any other fds on this mount can get mount info from the list.

This works because we have a valid fd with a mount point not listed in `/proc/$pid/mountinfo` so the fd has to belong to a detached mount.

3.3 Marking the mount as detached and storing details

We can add an optional `bool is_detached` field to `mnt.proto`'s `mnt_entry` message. We can store data from `statmount` into the existing `mnt_entry` message. Also, since mount namespaces are dumped after files when we add to the `mntinfo` list the mount should automatically get dumped.

3.4 Restoring

3.4.1 Creating a Temporary Bind Mount

We will have all the information we need to create a bind mount from the image files. We should do this before opening fds, i.e., alongside creating other mount points.

3.4.2 Lazily Unmounting These Mount Points

We can keep track of all mount points that will be lazily unmounted in a list. After all files have been opened, we can call `umount` with `MNT_DETACH` on each of the mount points.

4 Timeline

4.1 May 8 - June 1: Community Bonding Period

I would like to spend 1-2 weeks familiarizing and/or re-familiarizing with CRIU's codebase and learning more about mounts and related concepts that may be required in this project. After that, we can get to coding.

4.2 June 2 - June 16

1. Write the simplest possible zdtm test case that fails.
2. Figure out what extra info may need to be dumped and modify proto files.
3. Write the code for getting info on detached mounts from `statx` and `statmount`.

4.3 June 17 - June 30

1. Ensure all mount info (of detached mount) and fds on the mount and dumped correctly.
2. Write more tests for testing more edge cases.

4.4 July 1 - July 14: Mid Term

1. Some buffer time here to account for any delays.
2. Open up a draft PR for comments.
3. Finish everything related to checkpointing before the mid-term review.

4.5 July 14 - July 28

1. Figure out and implement the restoring of fds on detached mounts.

4.6 July 29 - August 28

1. Act on all feedback given in the PR and hopefully get the changes merged before the end of GSOC.
2. Write the final work report.

These timelines are rough estimates; some parts of the project may take more time than others. There's a lot still left to figure out, hence guidelines are intentionally vague.

5 Personal Information

- **Name:** Bhavik Sachdev
- **Email:** b.sachdev1904@gmail.com
- **GitHub:** bsach64
- **LinkedIn:** Bhavik Sachdev
- **Location:** Raipur, India
- **Timezone:** GMT +0530

5.1 About Me

I am currently a third-year student at IIIT Naya Raipur in India, pursuing a degree in Computer Science. I am also a part-time backend intern at Fam primarily working with their notification service.

5.2 Open Source Activity

I was part of Google Summer of Code 2024 with CRIU where I worked on `pidfd` support. Here's a brief summary of the work I have done with CRIU:

1. zdtm: Distinguish between fail and crash of dump
2. criu: Add support for pidfds
3. pidfd: block SIGCHLD during tmp process creation
4. Contributed a test case to fix this issue relating to dead pidfd restore.
5. Wrote a document describing checkpoint/restore for pidfd.

5.3 Commitments During GSOC 2025

I will finish my semester by 20th May 2025 and have no school commitments after that point. If Fam offers to extend my internship, I may continue working with them. However, I will dedicate the time needed to complete this project and am willing to extend the timeline if necessary.

6 References

1. manpage for mount
2. manpage for umount
3. manpage for mount_namespace
4. manpage for listmount
5. manpage for statmount
6. manpage for statx
7. Great sample on how to use `listmount` and `statmount`
8. Great article on the /dev directory