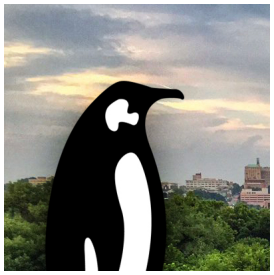

Contribution ID: **127** Type: **not specified**

Simplify Livepatch Callbacks, Shadow Variables, and States handling

Wednesday, 15 November 2023 11:30 (30 minutes)

Livepatches allow fixing critical security or functional bugs without reboot. They are useful when an down-time is expensive.

The basic livepatch feature functionality is to redirect problematic functions to fixed or improved variants. In addition, there are two features helping with more problematic situations:

- *pre_patch(), post_patch(), pre_unpatch(), post_unpatch()* **callbacks** might be called. For example, they allow to enable some functionality at the end of the transition when the entire system is using the new implementation. [1]
- **Shadow variables** allow to add new items into structures. [2]

Many fixes might be accumulated before the system gets rebooted. They might be added by separate livepatches or the existing fixes might be replaced by a **cumulative livepatch** including the old and new fixes.

The cumulative livepatches help with keeping the kernel in a well known state. All the changes are done by a single livepatch. It is easy especially when the livepatch modifies only the implementation of the patched functions.

The situation gets more complicated when callbacks and/or shadow variables are used. The new cumulative livepatch should **not do** some actions when they were **already done** by the previous livepatch. On the other side, it should **revert** some actions or **free** shadow variables when they are **no longer supported** by the new livepatch.

The problems with callbacks and shadow variables were supposed to be solved by **livepatch states** [3]. The API allows to check when the previous patch already had the state and behave accordingly. The states are versioned which allows to check if the new livepatch is compatible with the current one. The new livepatch could be installed only when it supports all the existing states.

The **practice shows** that all the pieces **do not play well** together:

- Shadow variables and states are **associated with the patch**.
- Callbacks are **associated with livepatched objects**. They are called when the livepatch or the livepatched module gets loaded or removed.
- **Only callbacks from the new livepatch** are called when it is replacing an older one which might prevent downgrades.
- The **state API is hard to use**.

But they **should be connected**. The callbacks are often used together with the shadow variables, for example:

- *post_patch()* callback might be need to enable using the shadow variable after the entire system is ready to handle them.
- *pre_unpatch()* callback might be needed to disable using the shadow variable.
- *post_unpatch()* callback might be needed to free the no longer used shadow variables.

The **shadow variables have a lifetime** [4]. They are introduced by one livepatch. They might be still used by newer patches. They need to get removed when the livepatch gets disabled or when they are no longer needed by a newer livepatch.

In fact, any changes done by the **callbacks have a lifetime**. It means that any state has a lifetime.

It is time to **connect** all the pieces a **better way**:

1. **Connect** callbacks and shadow variables with states.
2. Define all three pieces either **per-patch or per-object**.
3. Call the callbacks when the state is **introduced and removed**.

Proposal:

- **Move callbacks** to struct klp_state
- **Rename callbacks** to *setup*(), *enable*(), *disable*(), *remove*() and call them when the state is introduced and removed.
- **Add *@is_shadow*** flag to connect the state with a shadow variable with the same @id
- **Add *@shadow_dtor()*** callback to struct state and use it for garbage collection of obsolete shadow variables.
- **Add *@block_disable*** flag to prevent disabling or downgrading the livepatch when the state can't be disabled.

Pros:

- All pieces **play well together**.
- **Naturally support lifetime** of changes done by callbacks and shadow variables.
- disable() and remove() callbacks might be called from the patch which supported the state. It **allows downgrade** to a livepatch which was not aware of the state.
- **Obsoleting and disabling** states is **handled the same way**.

Cons:

- API changes are **not backward compatible**.
- Callback might be **called only when** the livepatch gets loaded when the state stay associated with the livepatch.
- **More complexity when** states get associated with livepatched object (modules).
- Callbacks are **not called when** the new livepatch support the same state. More callbacks might be needed when a **transition** is needed. Solvable by passing data via shadow variables?

Reference:

[1] https://lore.kernel.org/lkml/1507921723-1996-1-git-send-email-joe.lawrence@redhat.com/
[2] https://lore.kernel.org/lkml/1504211861-19899-1-git-send-email-joe.lawrence@redhat.com/
[3] https://lore.kernel.org/all/20191030154313.13263-3-pmladek@suse.com/T/#mf86ded54e03bf2a80a48d66040c381c9af219d89
[4] https://lore.kernel.org/all/20221026194122.11761-1-mpdesouza@suse.com/

Primary author: MLADEK, Petr (SUSE)

Presenter: MLADEK, Petr (SUSE)

Session Classification: Live Patching MC

Track Classification: LPC Microconference: Live Patching MC