ELISA
Enabling Linux in Safety Applications

WORKSHOP

Introducing SW Requirements in the Linux kernel development process: status and next steps

November 18-20, 2025
Co-hosted with Red Hat

Gabriele Paoloni (Red Hat)
Kate Stewart (Linux Foundation)
Chuck Wolber (Boeing)

Agenda

- Past Activities
- Feedbacks from the community
- Current activities
- Challenges and next steps

Past Activities: Evaluating and improving the Linux Kernel documentation

Main goals of the document:

- Analyze the current templates and guidelines that are available in the Linux Kernel documentation,
- Evaluate if and how they fulfill architecture and design aspects required by functional safety standards,
- Define improvements also in consideration of maintenance challenges deriving from a continuously evolving code baseline

This document triggered a session presented at Linux Plumbers 2024, following such a discussion the audience realized that we need to formalize and define testable requirements in Linux

Past Activities: Linux Kernel Requirements

Tag Name	Cardinality	Argument Mutability	Locations
SPDX-Req-ID	(1,1)	Immutable	Inline, Sidecar
SPDX-Req-End	(1,1)	N/A	Inline
SPDX-Req-Ref	(0,*)	Immutable	Inline
SPDX-Req-HKey	(1,1)	Mutable	Sidecar
SPDX-Req-Child	(0,*)	Mutable	Sidecar
SPDX-Req-Sys	(1,1)	Mutable	Sidecar
SPDX-Req-Text	(1,1)	Mutable	Sidecar
SPDX-Req-Note	(0,1)	Mutable	Sidecar

During the 2024 ELISA Workshop at NASA a requirements template proposal has been presented and we decided to prototype some examples and the automation associated with their generation and maintenance

Past Activities: Prototyping Linux Kernel Requirements

```
1498    /**
1499     * SPDX-Req-ID: [TODO automatically generate it]
1500     * SPDX-Req-Text:
1501     * trace_set_clr_event - enable or disable an event within a system
1502     * @system: system name (NULL for any system)
1503     * @event: event name (NULL for all events, within system)
1504     * @set: 1 to enable, 0 to disable (any other value is invalid)
1505     *
1506     * This is a way for other parts of the kernel to enable or disable
1507     * event recording.
1508     *
1509     * sequence of events:
1510     * 1) retrieve the global tracer
1511     * 2) locks the global event_mutex
1512     * 3) invokes __ftrace_set_clr_event_nolock
1513     * 4) unlocks the global event_mutex
1514     *
1515     * Returns 0 on success, -ENODEV if the global tracer cannot be retrieved,
1516     * -EINVAL if the parameters do not match any registered events, any other
1517     * error condition returned by __ftrace_set_clr_event_nolock
1518     */
1519    int trace_set_clr_event(const char *system, const char *event, int set)
1520    {
```

We started prototyping requirements for some functions of the tracing subsystem.
Requirements shall be:

- Testable
- Maintainable inline within the source code
- Compatible with pre-existing Kernel Documentation
- Hierarchically traceable

The main challenge is identifying the main design elements to be documented starting from the pre-existing code

While it is important to refer to design elements in the requirements (in order to write testable requirements), on the other hand it is not possible/feasible to mention them all (otherwise requirements would be as complex as the code itself)

Past Activities: Prototyping the automation to check patchsets against requirements

✓ **scripts/reqs/idgen.py: Add script for SPDX-Req-ID management"**

```
This script scans and processes all .c and .h files within a directory
tree.
It performs two main tasks:
* Preprocessing: Detects existing SPDX-Req-ID entries, updating a
  global map to track the highest progressive ID per file hash.
* ID Assignment: Updates or assigns new SPDX-Req-ID identifiers where
  missing, based on the file's hash and the next available progressive
  ID.

The script ensures efficient directory traversal by maintaining a single
file system scan and processes files in place, emitting warnings for
invalid or mismatched IDs.

Signed-off-by: Alessandro Carminati <acarmina@redhat.com>
```

alessandrocarminati committed 5 days ago

A script to automate the generation of Requirements' IDs (`SPDX-Req-ID`) is in-progress.

The goal is to generate a unique one that cannot change along the life of the requirements

"`SPDX-Req-HKey`" will instead be used to flag if, following code changes or requirement's text changes, the requirement shall be reviewed against the code (and vice versa).
"`SPDX-Req-HKey`" hashes are produced based on the following criteria:

- PROJECT: The name of the project (e.g. linux)
- FILE_PATH: The file the code resides in, relative to the root of the project repository.
- INSTANCE: The requirement template instance, minus tags with hash strings.
- CODE: The code that the SPDX-Req applies to.

"`SPDX-Req-ID`" is the very first "`SPDX-Req-HKey`" generated

Past Activities: Focusing on the design specifications

```
+/**
+ * mmap_mem - map physical memory into user space (/dev/mem).
+ * @file: file structure for the device.
+ * @vma: virtual memory area structure describing the user mapping.
+ *
+ * This function checks if the requested physical memory range is valid
+ * and accessible by the user, then it maps the physical memory range to
+ * user-mode address space.
+ *
+ * Function's expectations:
+ * 1. This function shall check if the requested physical address range to be
+ *    mapped fits within the maximum addressable physical range;
+ *
+ * 2. This function shall check if the requested  physical range corresponds to
+ *    a valid physical range and if access is allowed on it (if config STRICT_DEVMEM
+ *    is not set, access is always allowed);
+ *
+ * 3. This function shall check if the input virtual memory area can be used for
+ *    a private mapping (always OK if there is an MMU);
+ *
+ * 4. This function shall set the virtual memory area operations to
+ *    &mmap_mem_ops;
+ *
+ * 5. This function shall establish a mapping between the user-space
+ *    virtual memory area described by vma and the physical memory
+ *    range specified by vma->vm_pgoff and size;
+ *
+ * Context: process context.
+ *
+ * Return:
+ * * 0 on success
+ * * %-EAGAIN - invalid or unsupported mapping requested (remap_pfn_range()
+ *   fails)
+ * * %-EINVAL - requested physical range to be mapped is not valid
+ * * %-EPERM - no permission to access the requested physical range
+ */
```

- Before introducing the requirements formalism upstream we need to show the value
- The idea is to propose code specifications first and then introduce the requirements' framework
- Different RFCs have been submitted for the TRACING subsystem and /dev/mem:
  - https://lore.kernel.org/all/20250821170419.70668-1-gpaoloni@redhat.com/
  - https://lore.kernel.org/all/20250814122141.109076-1-gpaoloni@redhat.com/
  - https://lore.kernel.org/all/20250814122206.109096-1-gpaoloni@redhat.com/

Feedbacks from the community

```
> The format used for the **Description** intends to define testable
> function's expectations and Assumptions of Use to be met by the
> user of the function.

Where is this "format" documented?  Who will be parsing it?
```

One of the initial feedbacks was the need document how to write code specifications. On top of this it was flagged that a sample patchset should also include tests associated with the functions being specified

Feedbacks from the community

Jonathan Corbet

What I still don't really understand is what is the *purpose* of this formalized text? What will be consuming it? You're asking for a fair amount of effort to write and maintain these descriptions; what's in it for the people who do that work?

How does an author determine whether the specifications they have written are correct, both gramatically and semantically?

Thanks,

jon

Greg KH

So step back, and tell us exactly what files and functions and apis are needed to be documented in this stilted and formal way, who exactly is going to be doing all of that work, and why we should even consider reviewing and accepting and most importantly, maintaining such a thing for the next 40+ years.

A guideline for writing testable specifications has been proposed together with tests linking to the specified functions.

However skepticism was raised, mainly related to: “why we need specifications”, “who is going to write them”, “how many APIs will be documented”

Current Activities

```
+ Why, How, When to apply this guideline
+ ---------------------------------------
+
+ Why.
```

```
+ * integrators/users would clearly know what to expect without going through
+   the complexity of code internals;
+ * developers can verify if their proposed modifications are coherent and
+   consistent with the intended behavior;
+ * testers can test the code against the documented intended behavior, hence
+   without the risk of mistaking a bug for a feature.
```

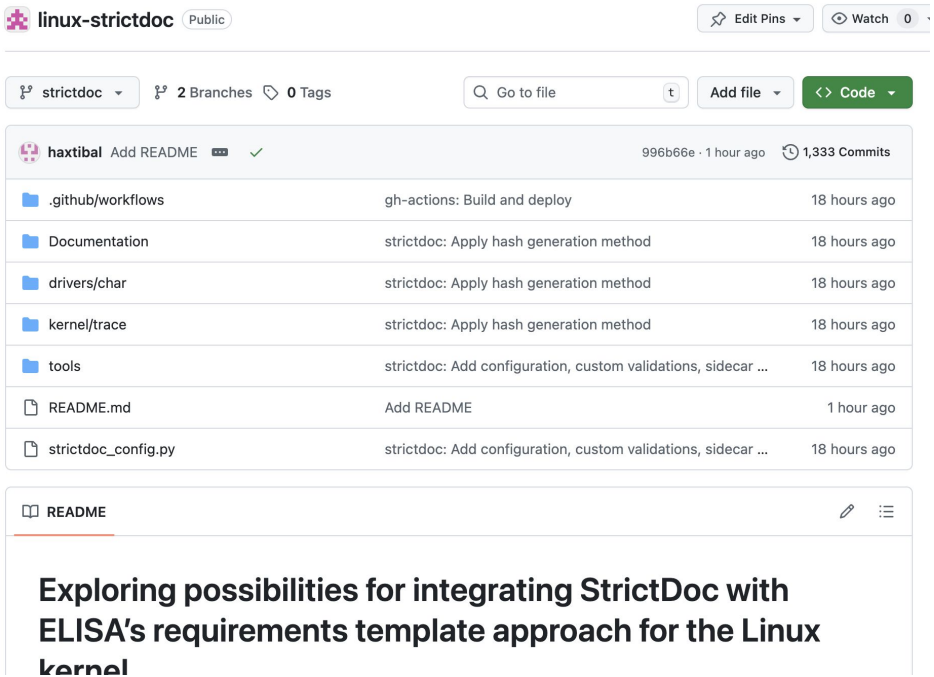
```
+ How
+ ~~~
```

```
+ In practical terms the individual contributors are expected to write such
+ testable specifications and the maintainers are expected to review them against
+ the pre-existing code and eventually accept them. Once upstream it should
+ be easier for new contributors and maintainers to verify code changes and also
+ for testers to write associated tests.
```

```
+ When
+ ~~~~
+ This guideline would be applied whenever individual contributors need to
+ specify the code intended behavior or whenever maintainers would like to do so
+ in order to reduce the technical debt associated with the maintained code.
```

We are addressing the skepticism from the community by revisiting the current guideline in preparation for the next patchset submission

Current Activities



While semantically we are working upstream on code specifications, syntactically we are experimenting the use of [strictdoc] as requirement processing tool.

The next session is dedicated to this activity

Next Steps

1) <<[RFC PATCH v3 0/3] Add testable code specifications>> will be submitted
2) Two sessions will be held at Linux Plumbers ‘25:
   a) Adding Testable Code Specifications in the Linux Kernel
   b) Defining and maintaining requirements in the Linux Kernel
3) In parallel we’ll continue working on the strictdoc integration

Licensing of Workshop Results

All work created during the workshop is licensed under Creative Commons Attribution 4.0 International (CC-BY-4.0) [https://creativecommons.org/licenses/by/4.0/] by default, or under another suitable open-source license, e.g., GPL-2.0 for kernel code contributions.

You are free to:

- Share — copy and redistribute the material in any medium or format
- Adapt — remix, transform, and build upon the material for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.


ELISA
Enabling Linux in
Safety Applications
WORKSHOP