Safety Architecture Working Group - Annual Update

Gabriele Paoloni & the WG participants

ELISA
Enabling Linux in Safety Applications

Aerospace · Automotive · Linux Features
Medical Devices · OS Engineering Process
Safety Architecture · Space Grade Linux · Systems · Tools

Agenda

- Working Group Intro
- Past years activities and milestones
- Current focus and activities
- What’s coming up in 2026 and areas and opportunities for collaboration
- Onboarding resources and how to get involved

The Safety Architecture Working Group

Mission:

According to technical safety requirements produced by domain specific WGs the focus of the Safety Architecture WG is to determine critical Linux subsystems and components in supporting safety functions, define associated safety requirements and scalable architectural assumptions, deliver corresponding safety analyses for their individual qualification and their integration into the safety critical system.

Past year activities

1) Linux Kernel Requirement Framework Definition
2) StrictDoc Integration
3) Upstreaming code specification and associated testing

LPC'24: "Improving kernel design documentation and involving experts"

Why Documenting the code is important

- New developers and security experts can verify if their proposed modifications are coherent and consistent with the intended behavior (e.g. when modifying the documented symbols).
- Reduction of incorrect or buggy patch proposals by new contributors thanks to faster understanding of the code behavior and side effects
- Testers can test the code against the documented intended behavior and documented constraints of use, hence without the risk of mistaking a bug for a feature. Moreover testing efficiency would be improved by more targeted testing around scope of impact of change.

In general the technical debt associated with Linux would be reduced.

The initial requirements' template

Tag Name	Cardinality	Argument Mutability	Locations	Description
SPDX-Req-ID	(1,1)	Immutable	Inline, Sidecar	Unique identifier for a requirement
SPDX-Req-Text	(1,1)	Mutable	Inline	semantic description of code expectations
SPDX-Req-End	(1,1)	N/A	Inline	End of requirement marker
SPDX-Req-Ref	(0,*)	Immutable	Inline	Marker for tagging code contributing to a certain requirement (e.g. for an inline helper function)
SPDX-Req-Note	(0,1)	Mutable	Inline	Any additional note needed for clarifications
SPDX-Req-HKey	(1,1)	Mutable	Sidecar	Hash key used to assess code or requirements' changes
SPDX-Req-Child	(0,*)	Mutable	Sidecar	identifier for enumerating lower level decomposed requirements
SPDX-Req-Sys	(1,1)	Mutable	Sidecar	subsystem identifier (as in running scripts/get_maintainer.pl)

StrictDoc tool

- Created in 2019, inspired by Doorstop
- Apache 2 license, 1.9K pull requests, 5K+ commits, 30K+ LOC
- In a nutshell:
  - **Let's cut prose and code into atomic nodes, give them UUIDs and attributes, and link them together to form a graph**
- Key highlights:
  - Connecting docs, requirements, source and test code, test reports, coverage.
  - Web-based requirements editor.
  - SDoc format for storing requirements with metadata. Internal representation is a graph.
  - Other formats can be read or written. Native ReqIF bi-directional interface.
  - RST export for interfacing with Sphinx. Possible direction: sphinx-strictdoc plugin.
  - Work with the SPDX FuSa WG. Establishing the equivalence between SPDX and SDoc.

RFC: Showcase on GitHub

Linux + ELISA + StrictDoc

https://github.com/strictdoc-project/linux-strictdoc

Results from CI:

rendered document

traceability graph

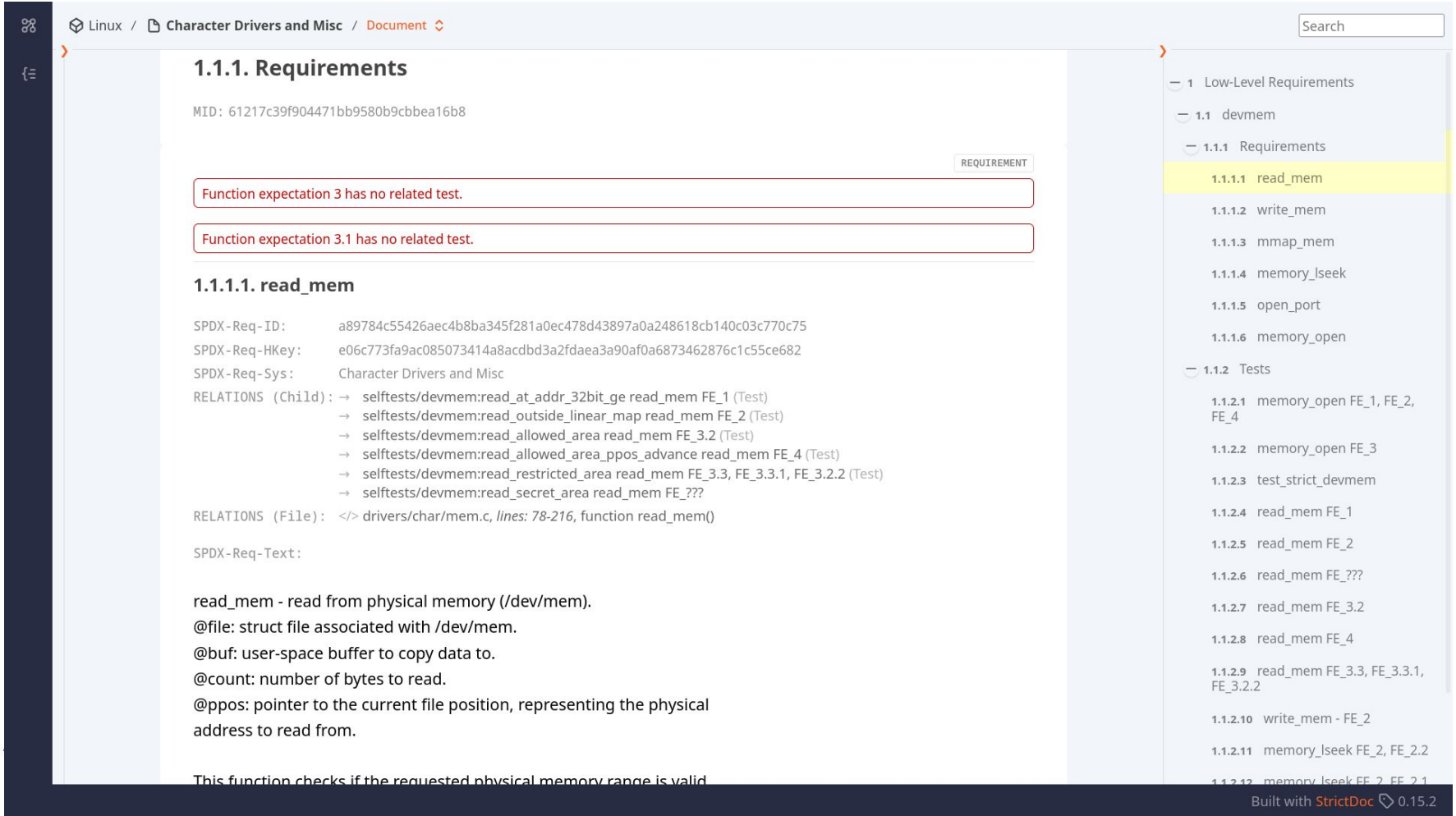
Input: Requirements from C Comments and Sidecar

```
/**
 * SPDX-Req-ID: 080fa9a6d27aa94dfaf8cbceb9715cbc146b0671bbe53c10dccf173f911b1a5e
 * SPDX-Req-Text:
 * trace_set_clr_event - enable or disable an event within a system.
 * @system: system name (NULL for any system).
 * [...]
 * Function's expectations:
 * - This function shall retrieve the pointer of the global trace array (global
 *   tracer) and pass it, along the rest of input parameters, to
 *   __ftrace_set_clr_event_nolock.
 * [...]
 * SPDX-Req-End
 */
int trace_set_clr_event(...) {...}
```

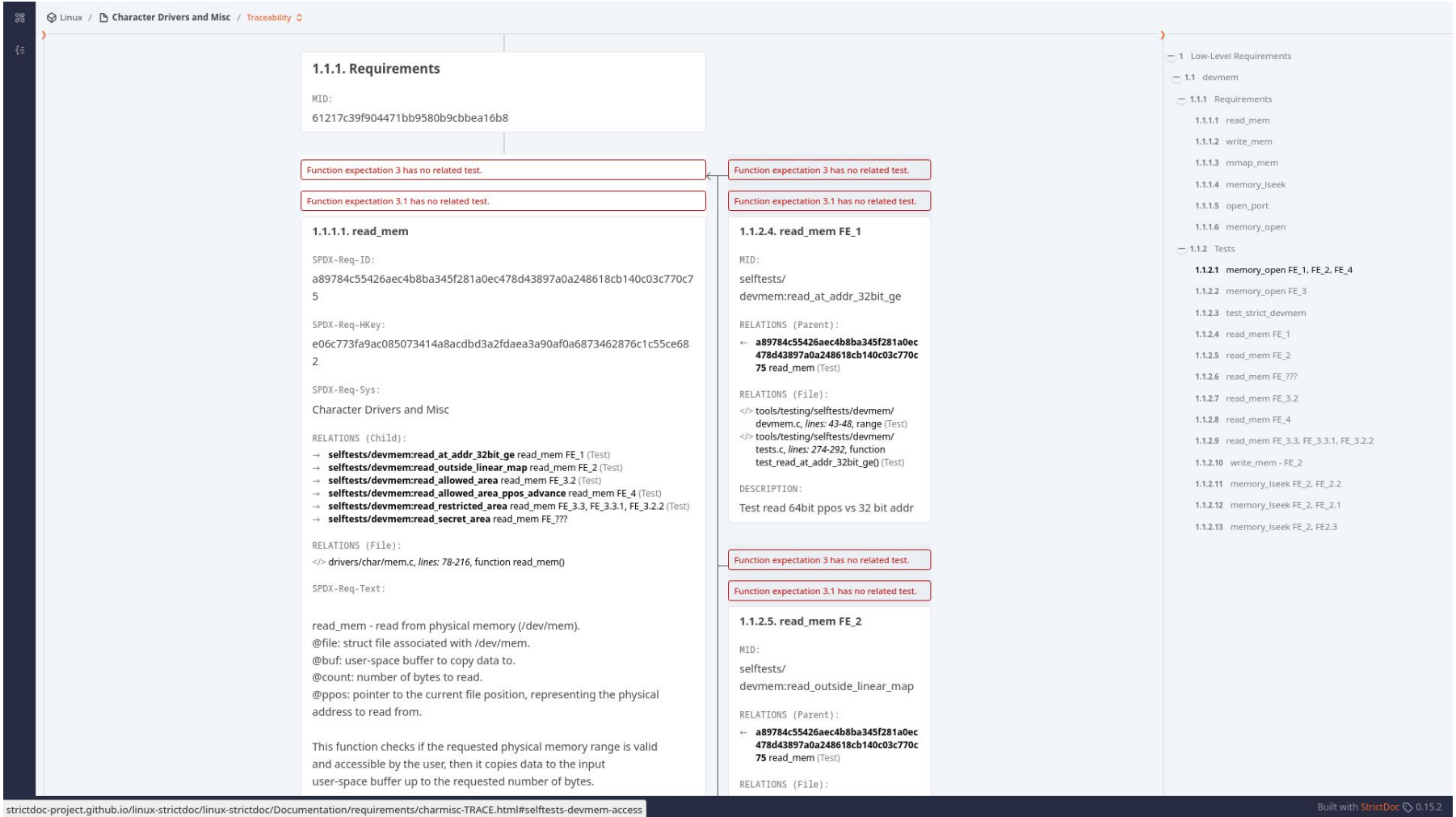

```
[REQUIREMENT]
MID: 080fa9a6d27aa94dfaf8cbceb9715cbc146b0671bbe53c10dccf173f911b1a5e
HASH: f8f29e7907a29e320df18a0950fa64b161dcd5cdd7960b44896e396bccb437c2
SPDX-Req-Sys: Tracing
TITLE: trace_set_clr_event
RELATIONS:
- TYPE: Parent
  VALUE: 428a5db6e481de87fc424119c30738d83e378b34bb42e12295ddfcba9839e5b3
```

(*) strictdoc export

Output: Document View with Links and Validations



Output: **Traceability View**



The initial requirement trials

```
/**
 * SPDX-Req-ID: [TODO automatically generate it]
 * SPDX-Req-Text:
 * trace_set_clr_event - enable or disable an event within a system.
 * @system: system name (NULL for any system).
 * @event: event name (NULL for all events, within system).
 * @set: 1 to enable, 0 to disable (any other value is invalid).
 *
 * This is a way for other parts of the kernel to enable or disable
 * event recording.
 *
 * Function's expectations:
 * - This function shall retrieve the pointer of the global trace array (global
 *   tracer) and pass it, along the rest of input parameters, to
 *   __ftrace_set_clr_event_nolock.
 * - This function shall properly lock/unlock the global event_mutex
 *   before/after invoking ftrace_set_clr_event_nolock.
 *
 * Returns 0 on success, -ENODEV if the global tracer cannot be retrieved,
 * -EINVAL if the parameters do not match any registered events, any other
 * error condition returned by __ftrace_set_clr_event_nolock.
 */
int trace_set_clr_event(const char *system, const char *event, int set)
{
        struct trace_array *tr = top_trace_array();

        if (!tr)
                return -ENODEV;

        return __ftrace_set_clr_event(tr, NULL, system, event, set, NULL);
}
EXPORT_SYMBOL_GPL(trace_set_clr_event);
```

The template was presented at the 2024 ELISA workshop hosted by NASA.

Positive feedbacks received by Steven Rostedt (TRACING subsystem maintainer).

The main suggestion was: “***before refining the syntax and formalism, work on the semantic aspects and show the value***”

TRACING subsystem's submissions

We mainly worked on "kernel/trace/trace_events.c" (see [1] and [2]) with **different outcomes**:

a bug was found (see [3])

```
From: Steven Rostedt <rostedt@goodmis.org>
To: linux-kernel@vger.kernel.org
Cc: Masami Hiramatsu <mhiramat@kernel.org>,
        Mark Rutland <mark.rutland@arm.com>,
        Mathieu Desnoyers <mathieu.desnoyers@efficios.com>,
        Andrew Morton <akpm@linux-foundation.org>,
        Gabriele Paoloni <gpaoloni@redhat.com>
Subject: [for-next][PATCH 08/10] tracing: fix return value in __ftrace_event_enable_disable for TRACE_REG_UNREGISTER
Date: Sun, 23 Mar 2025 08:29:41 -0400   [thread overview]
Message-ID: <20250323122950.397439862@goodmis.org> (raw)
In-Reply-To: 20250323122933.407277911@goodmis.org

From: Gabriele Paoloni <gpaoloni@redhat.com>

When __ftrace_event_enable_disable invokes the class callback to
unregister the event, the return value is not reported up to the
caller, hence leading to event unregister failures being silently
ignored.

This patch assigns the ret variable to the invocation of the
event unregister callback, so that its return value is stored
and reported to the caller, and it raises a warning in case
of error.

Link: https://lore.kernel.org/20250321170821.101403-1-gpaoloni@redhat.com
Signed-off-by: Gabriele Paoloni <gpaoloni@redhat.com>
Signed-off-by: Steven Rostedt (Google) <rostedt@goodmis.org>
```

[1] https://lore.kernel.org/all/20250814122206.109096-1-gpaoloni@redhat.com/
[2] https://lore.kernel.org/linux-trace-kernel/20250814122141.109076-1-gpaoloni@redhat.com/
[3] https://lore.kernel.org/all/20250323122950.397439862@goodmis.org/

TRACING subsystem's submissions

We mainly worked on "kernel/trace/trace_events.c" (see [1] and [2]) with **different outcomes**:

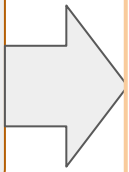
A redundant flag (SOFT_MODE) was identified and removed, with the code reworked and optimised (see [4])

```
From: Steven Rostedt <rostedt@kernel.org>
To: linux-kernel@vger.kernel.org
Cc: Masami Hiramatsu <mhiramat@kernel.org>,
        Mark Rutland <mark.rutland@arm.com>,
        Mathieu Desnoyers <mathieu.desnoyers@efficios.com>,
        Andrew Morton <akpm@linux-foundation.org>,
        Gabriele Paoloni <gpaoloni@redhat.com>
Subject: [for-next][PATCH 6/8] tracing: Remove EVENT_FILE_FL_SOFT_MODE flag
Date: Wed, 23 Jul 2025 10:49:13 -0400   [thread overview]
Message-ID: <20250723144928.341184323@kernel.org> (raw)
In-Reply-To: 20250723144907.219256132@kernel.org

From: Steven Rostedt <rostedt@goodmis.org>

When soft disabling of trace events was first created, it needed to have a
way to know if a file had a user that was using it with soft disabled (for
triggers that need to enable or disable events from a context that can not
really enable or disable the event, it would set SOFT_DISABLED to state it
is disabled). The flag SOFT_MODE was used to denote that an event had a
user that would enable or disable it via the SOFT_DISABLED flag.

Commit 1cf4c0732db3c ("tracing: Modify soft-mode only if there's no other
referrer") fixed a bug where if two users were using the SOFT_DISABLED
flag the accounting would get messed up as the SOFT_MODE flag could only
handle one user. That commit added the sm_ref counter which kept track of
how many users were using the event in "soft mode". This made the
SOFT_MODE flag redundant as it should only be set if the sm_ref counter is
non zero.

Remove the SOFT_MODE flag and just use the sm_ref counter to know the
event is in soft mode or not. This makes the code a bit simpler.

Link: https://lore.kernel.org/all/20250702111908.03759998@batman.local.home/

Cc: Masami Hiramatsu <mhiramat@kernel.org>
Cc: Mathieu Desnoyers <mathieu.desnoyers@efficios.com>
Cc: Gabriele Paoloni <gpaoloni@redhat.com>
Link: https://lore.kernel.org/20250702143657.18dd1882@batman.local.home
Signed-off-by: Steven Rostedt (Google) <rostedt@goodmis.org>
```


[1] https://lore.kernel.org/all/20250814122206.109096-1-gpaoloni@redhat.com/
[2] https://lore.kernel.org/linux-trace-kernel/20250814122141.109076-1-gpaoloni@redhat.com/
[3] https://lore.kernel.org/all/20250323122950.397439862@goodmis.org/
[4] https://lore.kernel.org/all/20250723144928.341184323@kernel.org/

/dev/mem submissions

We submitted a patchset with documentations, testable expectations and associated tests (see [5])



Subject: [RFC PATCH v2 0/3] Add testable code specifications
Date: Wed, 10 Sep 2025 18:59:57 +0200 [thread overview]
Message-ID: <20250910170000.6475-1-gpaoloni@redhat.com> (raw)

[1] was an initial proposal defining testable code specifications for some functions in /drivers/char/mem.c.
However a Guideline to write such specifications was missing and test cases tracing to such specifications were missing.
This patchset represents a next step and is organised as follows:
- patch 1/3 contains the Guideline for writing code specifications
- patch 2/3 contains examples of code specfications defined for some functions of drivers/char/mem.c
- patch 3/3 contains examples of selftests that map to some code specifications of patch 2/3

[1] https://lore.kernel.org/all/20250821170419.70668-1-gpaoloni@redhat.com/

Changes from v1:
1) Added a Guideline to write code specifications in the Linux Kernel Documentation
2) Addressed Greg KH comments in /drivers/char/mem.c
3) Added example of test cases mapping to the code specifications in /drivers/char/mem.c
---

> 1) In the first part of patch 1/3 we explain why we are doing this and the high
> level goals. Do you agree with these? Are these clear?

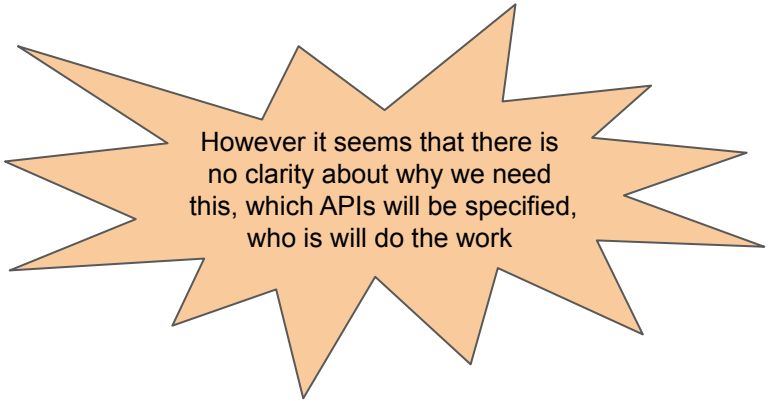
No, and no.

I think this type of thing is, sadly, folly. You are entering into a path that never ends with no clear goal that you are conveying here to us.

I might be totally wrong, but I fail to see what you want to have happen in the end.

Every in-kernel api documented in a "formal" way like this? Or a subset? If a subset, which ones specifically? How many? And who is going to do that? And who is going to maintain it? And most importantly, why is it needed at all?

For some reason Linux has succeeded in pretty much every place an operating system is needed for cpus that it can run on (zephyr for those others that it can not.) So why are we suddenly now, after many decades, requiring basic user/kernel stuff to be formally documented like this?



[5] https://lore.kernel.org/all/20250910170000.6475-1-gpaoloni@redhat.com/

Feedbacks received from maintainers at Linux Plumbers 2025

Greg KH on /dev/mem patchset: remove a the formal guideline on how to write requirements and submit code specifications and tests according to the current kernel-doc and KUnit formalisms

Discussion with other maintainers:

- There is a concern about the effort of maintaining low level specs and tests VS the real gain from introducing them;
- Specifications written in natural language could be interpreted differently by different contributors depending on the language proficiency and personal interpretation of the same;
- Kernel API tests may not be sufficient and should be complemented by architecture level tests (e.g. introducing formal models)

Current Focus

The top priorities are:

1) Keep working on upstream contributions (/dev/mem and TRACING subsystems in the very short term)
2) Show value in introducing requirements/specifications and testing (i.e. find and fix bugs)
3) Define a maintainable framework and associated development process

What’s coming up in 2026

- Get the current upstream RFCs accepted
- Finding new subsystems that may be more bug prone and show bug reduction for these when implementing requirements and tests
- Keep working on the StrictDoc integration to have a maintainable framework
- Start defining a requirement development and maintenance workflow for the Linux Kernel upstream process

Onboarding resources and how to get involved

WG Webpage: https://lists.elisa.tech/g/safety-architecture

WG Regular Public Meeting
All ELISA public meetings can be accessed here https://zoom-lfx.platform.linuxfoundation.org/meetings/elisa
You can register for a specific WG meeting to receive the direct meeting calendar invitation.
You can also subscribe to calendar feed here https://lists.elisa.tech/g/safety-architecture/calendar

GitHub Repo
Please go to https://github.com/elisa-tech/Safety_Architecture_WG for additional details including current work led by this group and how to collaborate.

ELISA
Enabling Linux in
Safety Applications

Thank You!