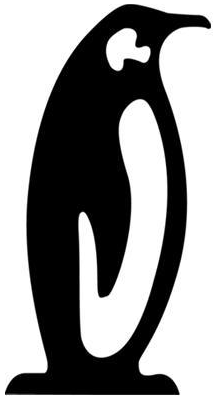


東京 2025 LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025

Defining and maintaining requirements in the Linux Kernel

Gabriele Paoloni
Senior Principal Software Engineer
Functional Safety
In-vehicle OS

Kate Stewart
Vice President
Dependable
Embedded Systems

Chuck Wolber
Associate Technical Fellow
Functional Safety
Airborne Software

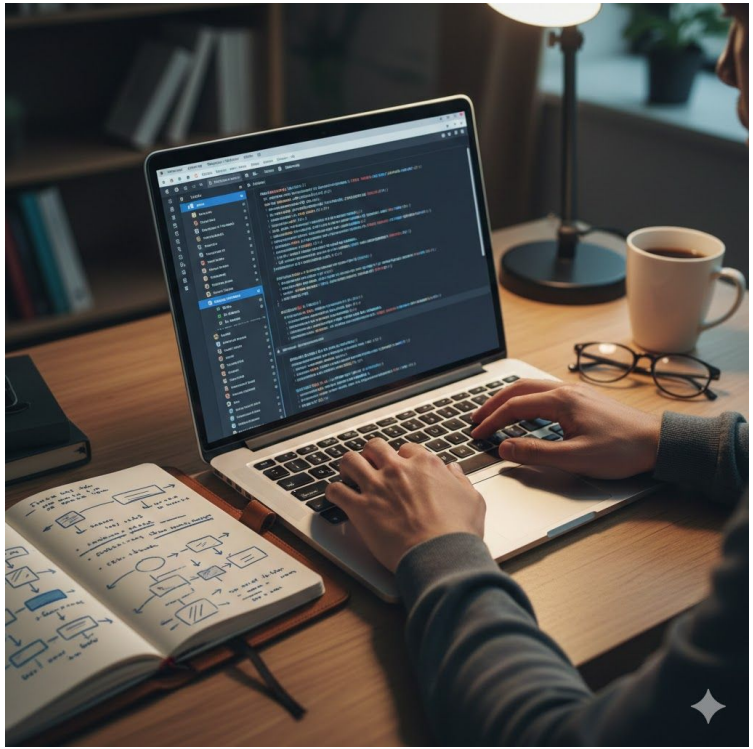
東京 2025
LINUX
PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

Agenda

- Initial Requirements' template
- Trials on the TRACING subsystem
- Trials on '/dev/mem'
- Open Discussion

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.

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

Outcome of the discussion

- Participating maintainers agreed that low level requirements are important to properly maintain code documentations;
- Participating maintainers asked to have semantic aspects of the requirements to be added inline with the source code whereas the rest of requirements' formalism should go into sidecar files;
- The requirements format must be compatible with the existing kernel documentation;
- The requirements' formalism should be supported by automation for evaluating requirements against newly submitted code.

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)

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/

TRACING subsystem's submissions

All of lore.kernel.org
search help / color / mirror / Atom feed

From: Gabriele Paoloni <gpaoloni@redhat.com>
To: rostedt@goodmis.org, mhiramat@kernel.org,
mathieu.desnoyers@efficios.com, linux-kernel@vger.kernel.org,
linux-trace-kernel@vger.kernel.org
Cc: acarmina@redhat.com, chuck.wolber@boeing.com,
Gabriele Paoloni <gpaoloni@redhat.com>
Subject: [RFC PATCH 1/2] tracing: fixes of ftrace_enable_fops
Date: Thu, 12 Jun 2025 12:43:48 +0200 [thread overview]
Message-ID: <20250612104349.5047-2-gpaoloni@redhat.com> (raw)
In-Reply-To: <20250612104349.5047-1-gpaoloni@redhat.com>

Currently there are different issues associated with ftrace_enable_fops
- event_enable_write: *ppos is increased while not used at all in the
write operation itself (following a write, this could lead a read to
fail or report a corrupted event status);
- event_enable_read: cnt < strlen(buf) is allowed and this can lead to
reading an incomplete event status (i.e. not all status characters
are retrieved) and/or reading the status in a non-atomic way (i.e.
the status could change between two consecutive reads);
- .llseek is set to default_llseek: this is wrong since for this
type of files it does not make sense to reposition the ppos offset.
Hence this should be set instead to noop_llseek.

This patch fixes all the issues listed above.

Signed-off-by: Gabriele Paoloni <gpaoloni@redhat.com>
Tested-by: Alessandro Carminati <acarmina@redhat.com>
---
kernel/trace/trace_events.c | 11 ++++++++---
1 file changed, 8 insertions(+), 3 deletions(-)

diff --git a/kernel/trace/trace_events.c b/kernel/trace/trace_events.c
index 120531268abf..5e84ef01d0c8 100644
--- a/kernel/trace/trace_events.c
+++ b/kernel/trace/trace_events.c
@@ -1798,6 +1798,13 @@ event_enable_read(struct file *filp, char __user *ubuf, size_

We realised that it is easy to mislead a feature for a bug (see [5]). So we introduced Assumptions of Use (AoUs) to document usage constraints

Just to confirm, I agree with Masami. The enable file is quite special, and I don't see the use of user space playing tricks with it, which even includes lseek. Maybe to keep rewinding a read to get a new status change?

But it usually contains a single character (sometimes two) and a new line. It's not something that's ever been reported as an issue. I rather not touch it if it hasn't been reported as broken because there's some hypothetical use case that can see it as broken.

Documenting its current behavior is perfectly fine with me.

-- Steve

[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/
[5] https://lore.kernel.org/all/20250612104349.5047-2-gpaoloni@redhat.com/

TRACING subsystem's submissions

```
+/**
+ * event_enable_read - read from a trace event file to retrieve its status.
+ * @filp: file pointer associated with the target trace event.
+ * @ubuf: user space buffer where the event status is copied to.
+ * @cnt: number of bytes to be copied to the user space buffer.
+ * @ppos: the current position in the buffer.
+ *
+ * This is a way for user space executables to retrieve the status of a
+ * specific event
+ *
+ * Function's expectations:
+ * - The global event_mutex shall be taken before performing any operation
+ *   on the target event;
+ *
+ * - The string copied to user space shall be formatted according to the
+ *   status flags from the target event file:
+ *   - If the enable flag is set AND the soft_disable flag is not set then
+ *     the first character shall be set to "1" ELSE it shall be set to "0";
+ *
+ *   - If either the soft_disable flag or the soft_mode flag is set then the
+ *     second character shall be set to "*" ELSE it is skipped;
+ *
+ *   - The string shall be terminated by a newline ("\n") and any remaining
+ *     character shall be set to "0";
+ *
+ * - This function shall invoke simple_read_from_buffer() to perform the copy
+ *   of the kernel space string to ubuf.
+ *
+ * Assumptions of Use:
+ * - The caller shall pass cnt equal or greater than the length of the string
+ *   to be copied to user space;
+ *
+ * - Any read operation on a file descriptor, unless it is the first operation
+ *   following a trace event file open, shall be preceded by an lseek
+ *   invocation to reposition the file offset to zero.
+ *
+ * Context: process context, locks and unlocks event_mutex.
+ *
+ * Return:
+ * * the number of copied bytes on success
+ * * %-ENODEV - the event file cannot be retrieved from the input filp
+ * * any error returned by simple_read_from_buffer
+ */
```

We realised that it is easy to mislead a feature for a bug (see [5]). So we introduced Assumptions of Use (AoUs) to document usage constraints

Just to confirm, I agree with Masami. The enable file is quite special, and I don't see the use of user space playing tricks with it, which even includes lseek. Maybe to keep rewinding a read to get a new status change?

But it usually contains a single character (sometimes two) and a new line. It's not something that's ever been reported as an issue. I rather not touch it if it hasn't been reported as broken because there's some hypothetical use case that can see it as broken.

Documenting its current behavior is perfectly fine with me.

-- Steve

[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/
[5] https://lore.kernel.org/all/20250612104349.5047-2-gpaoloni@redhat.com/

/dev/mem submissions

We proposed initial testable expectations for memory_open()
and most of the functions in the mem_fops structure (see [6])

```
From: Gabriele Paoloni <gpaoloni@redhat.com>
To: arnd@arndb.de, gregkh@linuxfoundation.org, linux-kernel@vger.kernel.org
Cc: safety-architecture@lists.elisa.tech,
        Gabriele Paoloni <gpaoloni@redhat.com>
Subject: [RFC PATCH] /dev/mem: Add initial documentation of memory_open() and mem_fops
Date: Thu, 21 Aug 2025 19:04:19 +0200   [thread overview]
Message-ID: <20250821170419.70668-1-gpaoloni@redhat.com> (raw)

This patch proposes initial kernel-doc documentation for memory_open()
and most of the functions in the mem_fops structure.
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.

Signed-off-by: Gabriele Paoloni <gpaoloni@redhat.com>
---
I have a couple of comments from this documentation activity:
1) Shouldn't the check in read_mem() <<if (p != *ppos)>> return
   -EFBIG (as done in write_mem())?
2) There is a note in memory_lseek() that states the return value
   to be (0) for negative addresses, however I cannot see how that
   would happen in the current implementation...
---
```

But we missed the guidelines and tests

```
> 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?
```

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

/dev/mem submissions

We submitted a new patchset with documentations, testable expectations and associated tests (see [7])

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
---

However it seems that there is no clarity about why we need this, which APIs will be specified, who is will do the work

> 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?

[6] https://lore.kernel.org/all/20250821170419.70668-1-gpaoloni@redhat.com/
[7] https://lore.kernel.org/all/20250910170000.6475-1-gpaoloni@redhat.com/

Open Discussion

Why are we doing this?

How many APIs (and ‘how’ in general)?

Who will do the work?

Open Discussion

Why are we doing this?

How many APIs (and ‘how’ in general)?

Who will do the work?

As per LPC2024 recap:

- 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).
- They would also understand the code behaviour and constraints for proper usage without going through the complexity of code internals (e.g. when writing code invoking the documented symbols).
- 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.

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

Open Discussion

Why are we doing this?

How many APIs (and ‘how’ in general)?

Who will do the work?

- The proposed guideline is an extension of the current kernel-doc guide that is fully compatible with the kernel-doc header format.
- The idea is to have a template for a more detailed specification of the code that can be used, if and when needed in association with corresponding tests. **We are not mandating a different way to document the code, just an optional improvement fully compatible with the current format**
- While it is difficult to estimate a number of APIs, our idea was to get some contributions upstream first. We expected maintainers to positively accept code that is specified more accurately with testing traced to such specifications as it should improve the quality of the code and help it become more bug resistant, and so reduce the maintainers' burden (as well as make it easier for new maintainers to adopt the code. Are there specific concerns?
- The introduction of the requirements’ formalism and automation would be the next steps after this intermediate semantic improvement

Open Discussion

Why are we doing this?

How many APIs (and ‘how’ in general)?

Who will do the work?

- Short Answer: the developers with vested interests in having the code documented and tested.
- Long Answer: as it happens today, kernel-doc headers documenting the code and associated tests are welcome by maintainers. A more accurate documentation and associated testing should reduce the maintainers’ burden (i.e. it should be easier to spot buggy submissions)

Any concerns that we missed?

Possible Solution

```
-
-High level goals
-----------------
-The code specifications:
-
-1. Should be maintainable together with the code.
-2. Should support hierarchical traceability to allow refinement of SW
-   dependencies (i.e. cross reference critical APIs or data structures).
-3. Should describe error conditions and success behaviors.
-4. Should describe conditions to be met by the user to avoid unspecified
-   or unwanted behaviors.
-5. Should allow covering both static and dynamic aspects of the code.
-6. Should be compatible with Documentation/doc-guide/kernel-doc.rst.
-7. Should support the definition of a test plan (i.e. syntax should enforce
-   testability as well as the avoidance of untestable specifications,
-   e.g “function_xyz() shall not do something”).
-
-Format and Syntax
-----------------
-Testable code specifications must be written according to the syntax already
-defined in Documentation/doc-guide/kernel-doc.rst with additional rules that
-are described below.
-
-Function name
-~~~~~~~~~~~~~
-``* function_name() - Brief description of function.``
-
-This field is to be considered informative and is not part of the testable
```

Step 1 -
Remove the documentation guideline

Possibly maintainers do not want to stick to a formalism to document the expected behaviour of the code and constraints of use

Possible Solution

```
gpaoloni@fedora:~/elisa_kernel/linux$ git diff Documentation/doc-guide/kernel-doc.
diff --git a/Documentation/doc-guide/kernel-doc.rst b/Documentation/doc-guide/kern
index af9697e60165..d90379b631fb 100644
--- a/Documentation/doc-guide/kernel-doc.rst
+++ b/Documentation/doc-guide/kernel-doc.rst
@@ -79,6 +79,25 @@ The general format of a function and function-like macro kernel
     * comment lines.
     *
     * The longer description may have multiple paragraphs.
+    *
+    * When specifying testable code behaviour the longer description must contain
+    * a paragraph formatted as follows:
+    *
+    * function_name beahviour:
+    * [ID1] - [expected behaviour]
+    *
+    * [ID2] - [expected behaviour]
+    *
+    * [...]
+    *
+    * [IDn] - [expected behaviour]
+    *
+    * function_name constraints of use:
+    * [ID1] - [constraint to be met by the caller]
+    *
+    * [ID2] - [constraint to be met by the caller]
+    *
+    * [IDn] - [constraint to be met by the caller]
     *
     * Context: Describes whether the function can sleep, what locks it takes,
```

Step 2 -
Define the minimal syntax constraints to allow traceability between test cases and code specifications

Make a small addition to kernel-doc.rst to define the syntax required to trace test cases to testable behaviour and to enumerate the constraints of use associated to be met by the caller

Possible Solution

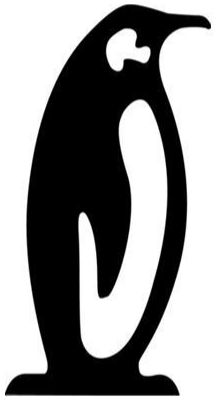
```
- * Function's expectations:
- *
- * 1. This function shall check if the value pointed by ppos exceeds the
- *    maximum addressable physical address;
- *
- * 2. This function shall check if the physical address range to be read
- *    is valid (i.e. it falls within a memory block and if it can be mapped
- *    to the kernel address space);
- *
- * 3. For each memory page falling in the requested physical range
- *    [ppos, ppos + count - 1]:
- *   3.1. this function shall check if user space access is allowed (if
- *        config STRICT_DEVMEM is not set, access is always granted);
- *
- *   3.2. if access is allowed, the memory content from the page range falling
- *        within the requested physical range shall be copied to the user space
- *        buffer;
- *
- *   3.3. zeros shall be copied to the user space buffer (for the page range
- *        falling within the requested physical range):
- *     3.3.1. if access to the memory page is restricted or,
- *     3.2.2. if the current page is page 0 on HW architectures where page 0 is
- *            not mapped.
- *
- * 4. The file position '*ppos' shall be advanced by the number of bytes
- *    successfully copied to user space (including zeros).
+ * read_mem behaviour:
+ * 1. it checks if the requested physical memory range [ppos, ppos + count - 1]
+ *    is valid;
+ * 2. for each page in the requested range, it checks if user space access is
+ *    allowed;
+ * 3. for each page in the requested range it copies data into the input
+ *    user-space buffer, padding with zero if access to the page is restricted
+ *    or the page is not mapped;
+ * 4. increases '*ppos' by the number of bytes successfully copied to user
+ *    space.
  *
```

Step 3 - Document the code according to the maintainer's guidance of the patched driver/subsystem

Maybe a more verbose and formal style fits the view of a certain maintainer whereas a different one prefers a less formal and more concise style.

However the less verbose specifications come with information loss for writing test cases

Any other ideas?



東京 2025
LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025