



LPC 2026: Defining Linux Kernel Requirements and Test Specifications out of the Kernel Tree

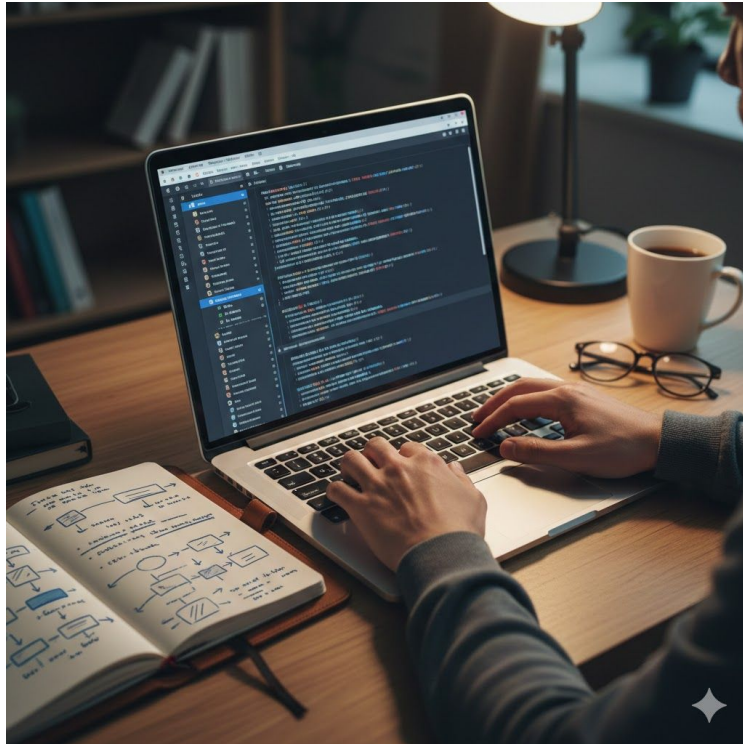
Gabriele Paoloni, Kate Stewart



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

Why Documenting the design and testing the code is important ?



Let's look at an example...

```
void drm_gem_shmem_vunmap_locked(struct drm_gem_shmem_object *shmem,
                                struct iosys_map *map)
{
    struct drm_gem_object *obj = &shmem->base;

    dma_resv_assert_held(obj->resv);

    if (drm_gem_is_imported(obj)) {
        dma_buf_vunmap(obj->import_attach->dmabuf, map);
    } else {
        dma_resv_assert_held(shmem->base.resv);

        if (refcount_dec_and_test(&shmem->vmap_use_count)) {
            vunmap(shmem->vaddr);
            shmem->vaddr = NULL;

            drm_gem_shmem_unpin_locked(shmem);
        }
    }
}
```

1st decision: is the object imported?

2nd decision: is the object still in use?

- Releases a virtual mapping of a SHMEM GEM object
- Two decisions, four leaf outcomes: imported or not, and the reference count reaching zero or staying positive.
- v7.3-rc5, drivers/gpu/drm/drm_gem_shmem_helper.c

The associated KUNIT test

```
static void drm_gem_shmem_test_vmap(struct kunit *test)
{
    struct drm_device *drm_dev = test->priv;
    struct drm_gem_shmem_object *shmem;
    struct iosys_map map;
    int ret, i;

    shmem = drm_gem_shmem_create(drm_dev, TEST_SIZE);
    KUNIT_ASSERT_NOT_ERR_OR_NULL(test, shmem);

    ret = kunit_add_action_or_reset(test, drm_gem_shmem_free_wrapper, shmem);
    KUNIT_ASSERT_EQ(test, ret, 0);

    ret = drm_gem_shmem_vmap(shmem, &map);
    KUNIT_ASSERT_EQ(test, ret, 0);
    KUNIT_ASSERT_NOT_NULL(test, shmem->vaddr);
    KUNIT_ASSERT_FALSE(test, iosys_map_is_null(&map));
    KUNIT_EXPECT_EQ(test, refcount_read(&shmem->vmap_use_count), 1);

    iosys_map_memset(&map, 0, TEST_BYTE, TEST_SIZE);
    for (i = 0; i < TEST_SIZE; i++)
        KUNIT_EXPECT_EQ(test, iosys_map_rd(&map, i, u8), TEST_BYTE);

    drm_gem_shmem_vunmap(shmem, &map);
    KUNIT_EXPECT_NULL(test, shmem->vaddr);
    KUNIT_EXPECT_EQ(test, refcount_read(&shmem->vmap_use_count), 0);
}
```

1st decision: is the object imported?
NO... it is created from scratch

2nd decision: is the object still in use?
NO...there only one vmap before
vunmap so the obj refcount reaches 0

— There is no dedicated vunmap test. The function is covered at the tail of a test whose stated objective is vmap, reached through the `drm_gem_shmem_vunmap()` wrapper that holds the reservation lock.

Two untested equivalence classes

```
void drm_gem_shmem_vunmap_locked(struct drm_gem_shmem_object *shmem,
                                struct iosys_map *map)
{
    struct drm_gem_object *obj = &shmem->base;

    dma_resv_assert_held(obj->resv);

    if (drm_gem_is_imported(obj)) {
        dma_buf_vunmap(obj->import_attach->dmabuf, map);
    } else {
        dma_resv_assert_held(shmem->base.resv);

        if (refcount_dec_and_test(&shmem->vmap_use_count)) {
            vunmap(shmem->vaddr);
            shmem->vaddr = NULL;

            drm_gem_shmem_unpin_locked(shmem);
        }
    }
}
```

1st decision: is the object imported?
NO

2nd decision: is the object still in
use? NO

the statement coverage
gap only detects the first
equivalence class missing

The untested equivalence classes

- 1st decision: yes, object imported
- 1st decision: no, not imported, 2nd decision: yes object still in use

The bug that made it through....

```
commit e1fc39a923329c34c61dfed0665620d21826e8f4
```

```
[...]
```

```
diff --git a/drivers/gpu/drm/drm_gem_shmem_helper.c b/drivers/gpu/drm/drm_gem_shmem_helper.c
```

```
index 84a196bbe44f..2d924d547a51 100644
```

```
--- a/drivers/gpu/drm/drm_gem_shmem_helper.c
```

```
+++ b/drivers/gpu/drm/drm_gem_shmem_helper.c
```

```
[...]
```

```
@@ -413,14 +413,10 @@ void drm_gem_shmem_vunmap_locked(struct drm_gem_shmem_object *shmem,
```

```
    } else {
```

```
        dma_resv_assert_held(shmem->base.resv);
```

```
-         if (drm_WARN_ON_ONCE(obj->dev, !shmem->vmap_use_count))
```

```
-             return;
```

```
-         if (--shmem->vmap_use_count > 0) ←
```

```
-             return;
```

```
-         vunmap(shmem->vaddr);
```

```
-         drm_gem_shmem_unpin_locked(shmem);
```

```
+         if (refcount_dec_and_test(&shmem->vmap_use_count)) {
```

```
+             vunmap(shmem->vaddr);
```

```
+             drm_gem_shmem_unpin_locked(shmem);
```

```
+         }
```

```
}
```

```
shmem->vaddr = NULL; ←
```

In the previous implementation if the object was still in use the function was returning right away

Now, even when the object is still in use, we set vaddr to NULL

The bug went undetected by KUNITs

```
void drm_gem_shmem_vunmap_locked(struct drm_gem_shmem_object *shmem,
                                struct iosys_map *map)
{
    struct drm_gem_object *obj = &shmem->base;

    dma_resv_assert_held(obj->resv);

    if (drm_gem_is_imported(obj)) {
        dma_buf_vunmap(obj->import_attach->dmabuf, map);
    } else {
        dma_resv_assert_held(shmem->base.resv);

        if (refcount_dec_and_test(&shmem->vmap_use_count)) {
            vunmap(shmem->vaddr);

            drm_gem_shmem_unpin_locked(shmem);
        }

        shmem->vaddr = NULL;
    }
}
```

1st decision: is the object imported?
NO

2nd decision: is the object still in
use? NO

**There is no test for the “object still in use” equivalence class verifying vaddr to be non-NULL.
If such a test had been in place probably the patch above would not have gone through**

Finally the bug was fixed

```
commit dd717762761807452ca25634652e180a80349cd8
```

```
[...]
```

```
Fixes: e1fc39a92332 ("drm/shmem-helper: Use refcount_t for vmap_use_count")
```

```
Reported-by: Lucas De Marchi <lucas.demarchi@intel.com>
```

```
Closes:
```

```
https://lore.kernel.org/dri-devel/20250403105053.788b0f6e@collabora.com/T/#m3dca6d81bedc8d6146a56b82694624fbc6fa4c96
```

```
Signed-off-by: Dmitry Osipenko <dmitry.osipenko@collabora.com>
```

```
Reviewed-by: Boris Brezillon <boris.brezillon@collabora.com>
```

```
Tested-by: Lucas De Marchi <lucas.demarchi@intel.com>
```

```
Link: https://lore.kernel.org/r/20250403142633.484660-1-dmitry.osipenko@collabora.com
```

```
diff --git a/drivers/gpu/drm/drm_gem_shmem_helper.c b/drivers/gpu/drm/drm_gem_shmem_helper.c
```

```
index 2d924d547a51..aa43265f4f4f 100644
```

```
--- a/drivers/gpu/drm/drm_gem_shmem_helper.c
```

```
+++ b/drivers/gpu/drm/drm_gem_shmem_helper.c
```

```
@@ -415,11 +415,11 @@ void drm_gem_shmem_vunmap_locked(struct drm_gem_shmem_object *shmem,
```

```
    if (refcount_dec_and_test(&shmem->vmap_use_count)) {
```

```
        vunmap(shmem->vaddr);
```

```
        shmem->vaddr = NULL;
```

```
        drm_gem_shmem_unpin_locked(shmem);
```

```
    }
```

```
}
```

```
- shmem->vaddr = NULL;
```

```
}
```

An extra note

v6.15 implementation

```
void drm_gem_shmem_vunmap(struct drm_gem_shmem_object *shmem,
                          struct iosys_map *map)
{
    struct drm_gem_object *obj = &shmem->base;

    if (drm_gem_is_imported(obj)) {
        dma_buf_vunmap(obj->dma_buf, map);
    } else {
        dma_resv_assert_held(shmem->base.resv);

        if (drm_WARN_ON_ONCE(obj->dev, !shmem->vmap_use_count))
            return;

        if (--shmem->vmap_use_count > 0)
            return;

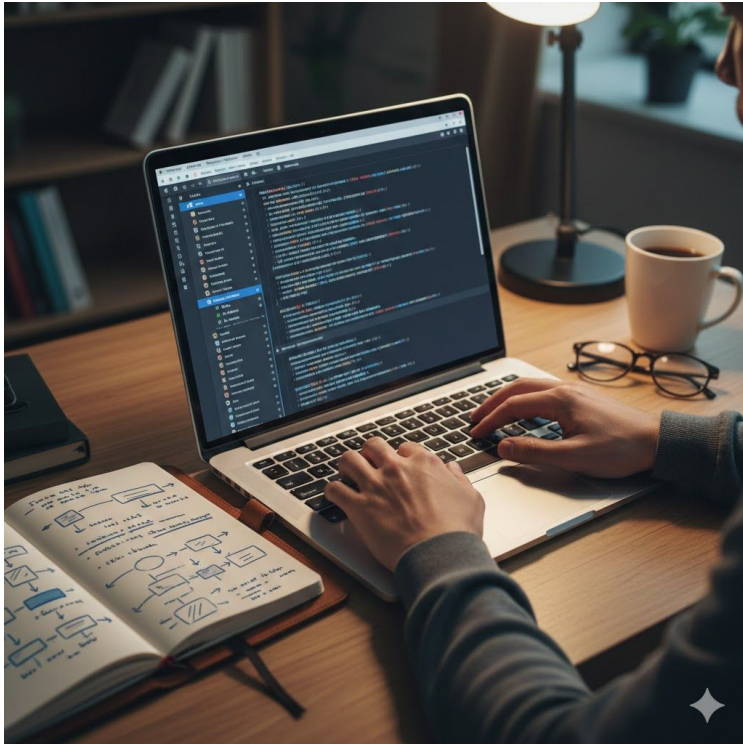
        vunmap(shmem->vaddr);
        drm_gem_shmem_put_pages(shmem);
    }

    shmem->vaddr = NULL;
}

EXPORT_SYMBOL(drm_gem_shmem_vunmap);
```

in the previous implementation vaddr is set to NULL also when the gem object is imported...not sure this is fully correct

Why Documenting the design and testing the code is important (also for `drm_gem_shmem_vunmap`) ?



- A new developer can check the behavioral expectations and make sure that his modifications meet them (e.g. `vaddr` set to `NULL` only when the shared object has no active users)
- This would lead to less incorrect or buggy proposals
- A tester can verify all the behavioral expectations to be tested (e.g. by writing new test specs and test cases for the use cases of imported objects or for a mapped object shared by multiple users)

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

However how do we maintain these set of collaterals associated with the code?

Initially we proposed a model for maintaining requirements, test specifications and test cases upstream with the code



LPC'24: “Improving kernel design documentation and involving experts”

Discussion outcome

- Participating maintainers agreed that low level requirements are important to properly uphold intended design;
- 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.

/dev/mem submissions

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

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

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

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

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 specifications 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.

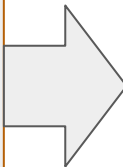
I might be totally wrong, but I fail to see what you want to have happen in the end.

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?

/dev/mem submissions

Following LPC25 discussions and feedbacks we proposed a further patchset with the minimum formalism to trace specifications to test cases (see [8])

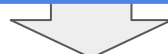


RFCv1 [1] was an initial proposal defining testable code specifications for some functions in /drivers/char/mem.c. However it was not clear how to write specifications according to the format and wording used for them, plus tests were missing.

RFCv2 [2] proposed a guideline explaining how to write such testable code specifications and also added kselftest for read_mem, however there was reluctance to accept the proposed addition before having clarity on who needs it, who is going to maintain it and for which APIs in the kernel it is needed.

RFCv3 (this one) proposes kernel-doc API documentation according to the current doc-guide for some APIs of /dev/mem introducing the concept of traceable specifications to KUnit tests.

For read_mem() Kunit tests traced to the corresponding specifications are also introduced, so that in case of test failure it is clear which specification is violated and it would be easier to review incoming patches against the intended behavior of the code.



Again this was Rejected.

```
> + * [ID2] - [constraint to be met by the caller]
> + *
> + * [IDn] - [constraint to be met by the caller]
```

So the same "id" is used for a behavior, AND a constraint?

And what defines an "id"? I see in your example you use number.number, but is that specified?

And how is a id going to stay in sync across different files? That feels impossible to maintain for any length of time, and puts a burden on the developer who wishes to add/remove a test or "id", AND a maintainer who has to remember to go and look in multiple places for such an id sync up.

That's just not going to work, sorry.

greg k-h

[8] <https://lore.kernel.org/all/20260212124923.222484-1-gpaoloni@redhat.com/>

New Idea: Maintaining Requirements and Test Specs in a separate repo



REQUIREMENT DATABASE			
REQ ID	DESCRIPTION	STATUS	OWNER
REQ-101	Define core function behavior for error cases	Active	JOHN SON
REQ-102	Security checks for critical data structures	Pending	ALICE LEE
REQ-103	Implement comprehensive success path criteria	Active	JOHN SON
REQ-104	Optimize performance for data processing loop	Verified	Stewart, Kate



TEST SPECIFICATIONS DATABASE				
TEST ID	TEST DESCRIPTION	TRACES TO REQ	TEST TYPE	OWNER
TST-001	Verify error path	REQ-001	Functionality	ANNE SMITH
TST-002	Mmap access test	REQ-002	Security	BOB CHEN
TST-202	Static analysis of critical data structures in /dev/mem	REQ-102	Inspection	BOB CHEN
TST-203	Compliance test for success path criteria	REQ-103	Analysis	Canonical
TST-204	Benchmarking of data processing optimization	REQ-104	Audit	ELISA team



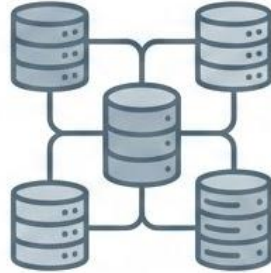
LINUX KERNEL REPOSITORY			
CODE FILE PATHS	FUNCTION/METHOD NAMES	COMMIT ID	DEVELOPER NAME
core.c	check_data_access code)	000013	ALICE LEE
security.c	process_data_stream (data_ptr)	000043	JOHN SON
data_processor.c	unroll_loop (stream_ptr)	000112	ALICE LEE
unmap	unmap	e1fc39a9	Osipenko, Dmitry

New Idea: Maintaining Requirements and Test Specs in a separate repo



ZERO IMPACT

There is no burden imposed
Requirements and Test
Specifications traceability in
development process and
hence on maintainers.



DECENTRALIZED REPOS

Requirements, test
specifications, test cases
can be maintained across
diverse repositories



TRACEABILITY OPT-IN

Maintainers can still look at
traceability gaps (toward
public repos) if they find it
valuable.

Example drm_gem_shmem_vunmap_locked

Completion:



```
void drm_gem_shmem_vunmap_locked(struct drm_gem_shmem_object *shmem,
                                struct iosys_map *map)
{
    struct drm_gem_object *obj = &shmem->base;

    dma_resv_assert_held(obj->resv);

    if (drm_gem_is_imported(obj)) {
        dma_buf_vunmap(obj->import_attach->dmabuf, map);
    } else {
        dma_resv_assert_held(shmem->base.resv);

        if (refcount_dec_and_test(&shmem->vmap_use_count)) {
            vunmap(shmem->vaddr);
            shmem->vaddr = NULL;

            drm_gem_shmem_unpin_locked(shmem);
        }
    }
}
```



Software Requirement 4



ver. 1.1

new

0.0% Completion

DRM-GEM-SHMEM-VUNMAP-LOCKED-001

The `drm_gem_shmem_vunmap_locked()` function shall:

1. Assert that the reservation lock of the base Graphics Execution Manager (GEM) object derived from the input `shmem` (representing the Shared Memory (SHMEM) GEM object) is held using `dma_resv_assert_held()`.
2. If the base GEM object is imported (determined by calling the static inline function `drm_gem_is_imported()`, which relies on the `import_attach` field of the base GEM object to identify imported objects):
 - 2.1. Unmap the virtual mapping of the Direct Memory Access (DMA) buffer associated with the imported attachment using `dma_buf_vunmap()`, passing the input `map`.
3. If the base GEM object is not imported:
 - 3.1. Assert that the reservation lock of the base GEM object derived from the input `shmem` is held using `dma_resv_assert_held()`.
 - 3.2. If decrementing the virtual mapping use count of the input `shmem` reaches zero (evaluated using `refcount_dec_and_test()`):
 - 3.2.1. Unmap the virtual address of the input `shmem` using `vunmap()`.
 - 3.2.2. Clear the virtual address pointer of the input `shmem`.
 - 3.2.3. Unpin the backing pages of the input `shmem` by calling `drm_gem_shmem_unpin_locked()`, which relies on the `pages_pin_count` field of the input `shmem` to conditionally release the pages.
4. Return no value.

Example drm_gem_shmem_vunmap_locked



Software Requirement 4

ver. 1.1

new

0.0% Completion

DRM-GEM-SHMEM-VUNMAP-LOCKED-001

The `drm_gem_shmem_vunmap_locked()` function shall:

1. Assert that the reservation lock of the base Graphics Execution Manager (GEM) object derived from the input `shmem` (representing the Shared Memory (SHMEM) GEM object) is held using `dma_resv_assert_held()`.
2. If the base GEM object is imported (determined by calling the static inline function `drm_gem_is_imported()`, which relies on the `import_attach` field of the base GEM object to identify imported objects):
 - 2.1. Unmap the virtual mapping of the Direct Memory Access (DMA) buffer associated with the imported attachment using `dma_buf_vunmap()`, passing the input `map`.
3. If the base GEM object is not imported:
 - 3.1. Assert that the reservation lock of the base GEM object derived from the input `shmem` is held using `dma_resv_assert_held()`.
 - 3.2. If decrementing the virtual mapping use count of the input `shmem` reaches zero (evaluated using `refcount_dec_and_test()`):
 - 3.2.1. Unmap the virtual address of the input `shmem` using `vunmap()`.
 - 3.2.2. Clear the virtual address pointer of the input `shmem`.
 - 3.2.3. Unpin the backing pages of the input `shmem` by calling `drm_gem_shmem_unpin_locked()`, which relies on the `pages_pin_count` field of the input `shmem` to conditionally release the pages.
4. Return no value.



Test Specification 4

ver. 1.1

new

0.0% Completion

DRM-GEM-SHMEM-VUNMAP-LOCKED-001

Preconditions:

- a. Target context: KUnit test module running on a Linux kernel built from the tree under test (v7.3-rc5). The device under test is a real DRM GEM shared memory object; no hardware is simulated and no test doubles are used.
- b. Kconfig: `CONFIG_DRM`, `CONFIG_DRM_GEM_SHMEM_HELPER`, `CONFIG_KUNIT` and `CONFIG_DRM_KUNIT_TEST_HELPERS` enabled.
- c. Platform/device state: a parent device from `drm_kunit_helper_alloc_device()` with a 64-bit DMA mask; a struct `drm_device` from `__drm_kunit_helper_alloc_drm_device()` stored in `test->priv`; a struct `drm_gem_shmem_object` created by `drm_gem_shmem_create(drm_dev, TEST_SIZE)`. The object is therefore not imported: `base.import_attach` is `NULL` as a property of creation, not as a value planted by the test. Its mapping state at creation is `vaddr == NULL` and `vmap_use_count == 0`.
- d. Mock and fault injection state: None. `dma_resv_assert_held()`, `vunmap()` and `drm_gem_shmem_unpin_locked()` are the real implementations, and the reservation lock is genuinely acquired and released.
- e. Cleanup-triggering state: the object is registered with `kunit_add_action_or_reset(drm_gem_shmem_free_wrapper)`, so `drm_gem_shmem_free()` runs at the end of the test.
- f. Global and file-scope state: None.
- g. Invocation route: `drm_gem_shmem_vunmap_locked()` is invoked indirectly, through the `drm_gem_shmem_vunmap()` wrapper, which holds the reservation lock across the call. The test does not call the locked helper directly, so the lock precondition in `SHALL 1` and `SHALL 3.1` is satisfied by production code rather than by test setup.
- h. Observability: only the object's own fields are observable – `shmem->vaddr` and

[...]

Example drm_gem_shmem_vunmap_locked

Test Specification 4

ver. 1.1 new 0.0% Completion

DRM-GEM-SHMEM-VUNMAP-LOCKED-001

Preconditions:

- Target context: KUnit test module running on a Linux kernel built from the tree under test (v7.3-rc5). The device under test is a real DRM GEM shared memory object; no hardware is simulated and no test doubles are used.
- Kconfig: CONFIG_DRM, CONFIG_DRM_GEM_SHMEM_HELPER, CONFIG_KUNIT and CONFIG_DRM_KUNIT_TEST_HELPERS enabled.
- Platform/device state: a parent device from `drm_kunit_helper_alloc_device()` with a 64-bit DMA mask; a struct `drm_device` from `__drm_kunit_helper_alloc_device()` stored in `test->priv`; a struct `drm_gem_shmem_object` created by `drm_gem_shmem_create(drm_dev, TEST_SIZE)`. The object is therefore not imported: `base.import_attach` is NULL as a property of creation, not as a value planted by the test. Its mapping state at creation is `vaddr == NULL` and `vmap_use_count == 0`.
- Mock and fault injection state: None. `dma_resv_assert_held()`, `vunmap()` and `drm_gem_shmem_unpin_locked()` are the real implementations, and the reservation lock is genuinely acquired and released.
- Cleanup-triggering state: the object is registered with `kunit_add_action_or_reset(drm_gem_shmem_free_wrapper)`, so `drm_gem_shmem_free()` runs at the end of the test.
- Global and file-scope state: None.
- Invocation route: `drm_gem_shmem_vunmap_locked()` is invoked indirectly, through the `drm_gem_shmem_vunmap()` wrapper, which holds the reservation lock across the call. The test does not call the locked helper directly, so the lock precondition in SHALL 1 and SHALL 3.1 is satisfied by production code rather than by test setup.
- Observability: only the object's own fields are observable — `shmem->vaddr` and

```
static void drm_gem_shmem_test_vmap(struct kunit *test)
{
    struct drm_device *drm_dev = test->priv;
    struct drm_gem_shmem_object *shmem;
    struct iosys_map map;
    int ret, i;

    shmem = drm_gem_shmem_create(drm_dev, TEST_SIZE);
    KUNIT_ASSERT_NOT_ERR_OR_NULL(test, shmem);

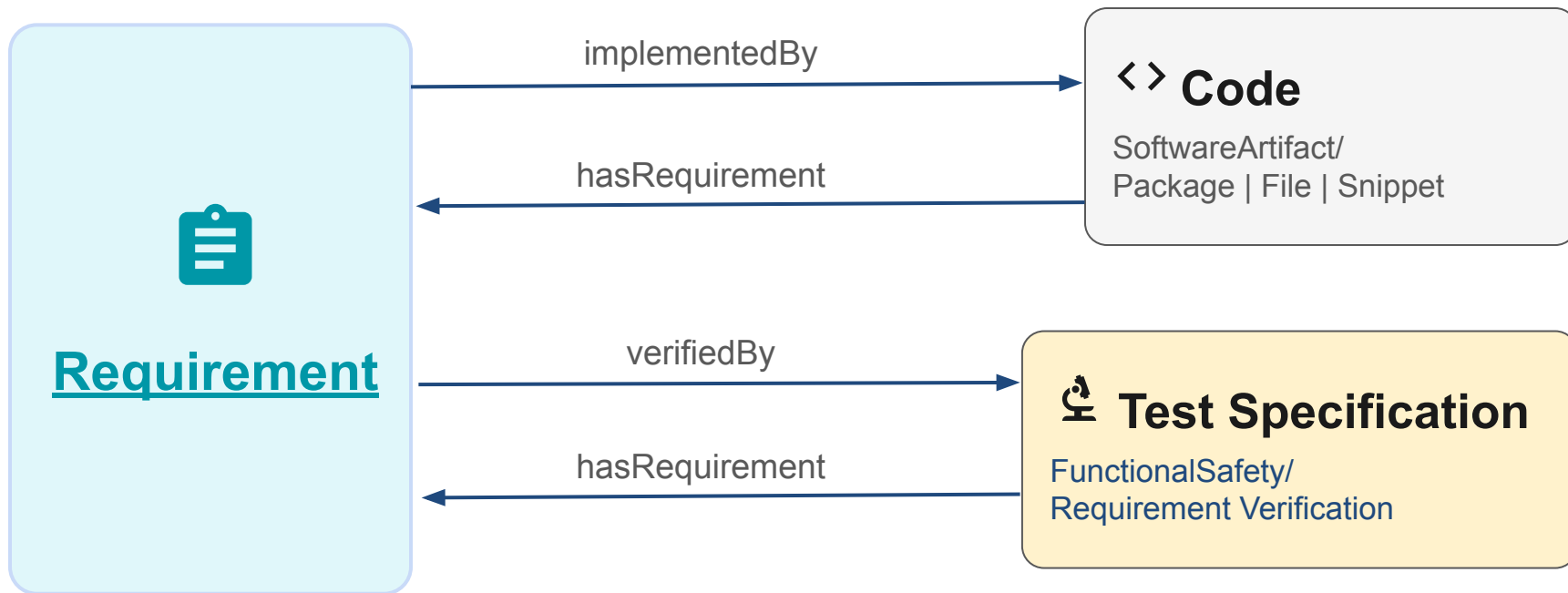
    ret = kunit_add_action_or_reset(test, drm_gem_shmem_free_wrapper, shmem);
    KUNIT_ASSERT_EQ(test, ret, 0);

    ret = drm_gem_shmem_vmap(shmem, &map);
    KUNIT_ASSERT_EQ(test, ret, 0);
    KUNIT_ASSERT_NOT_NULL(test, shmem->vaddr);
    KUNIT_ASSERT_FALSE(test, iosys_map_is_null(&map));
    KUNIT_EXPECT_EQ(test, refcount_read(&shmem->vmap_use_count), 1);

    iosys_map_memset(&map, 0, TEST_BYTE, TEST_SIZE);
    for (i = 0; i < TEST_SIZE; i++)
        KUNIT_EXPECT_EQ(test, iosys_map_rd(&map, i, u8), TEST_BYTE);

    drm_gem_shmem_vunmap(shmem, &map);
    KUNIT_EXPECT_NULL(test, shmem->vaddr);
    KUNIT_EXPECT_EQ(test, refcount_read(&shmem->vmap_use_count), 0);
}
```

SPDX 3.1-dev (active development)



Requirement (active development)

Field Purpose	Cardinality	SPDX 3.1
Identify Requirement	(1,1)	core/ spdxID core/Requirement/ requirementUID
Requirement Statement	(1,1)	core/Requirement/ requirementStatement
Requirement Integrity	(1,1)	core/requirement/ verifiedUsing (see Integrity Method for options)
Rationale	(0,1)	core/Requirement/ rationale

Under discussion:

requirement state/status - DRAFT | APPROVED | DEPRECATED

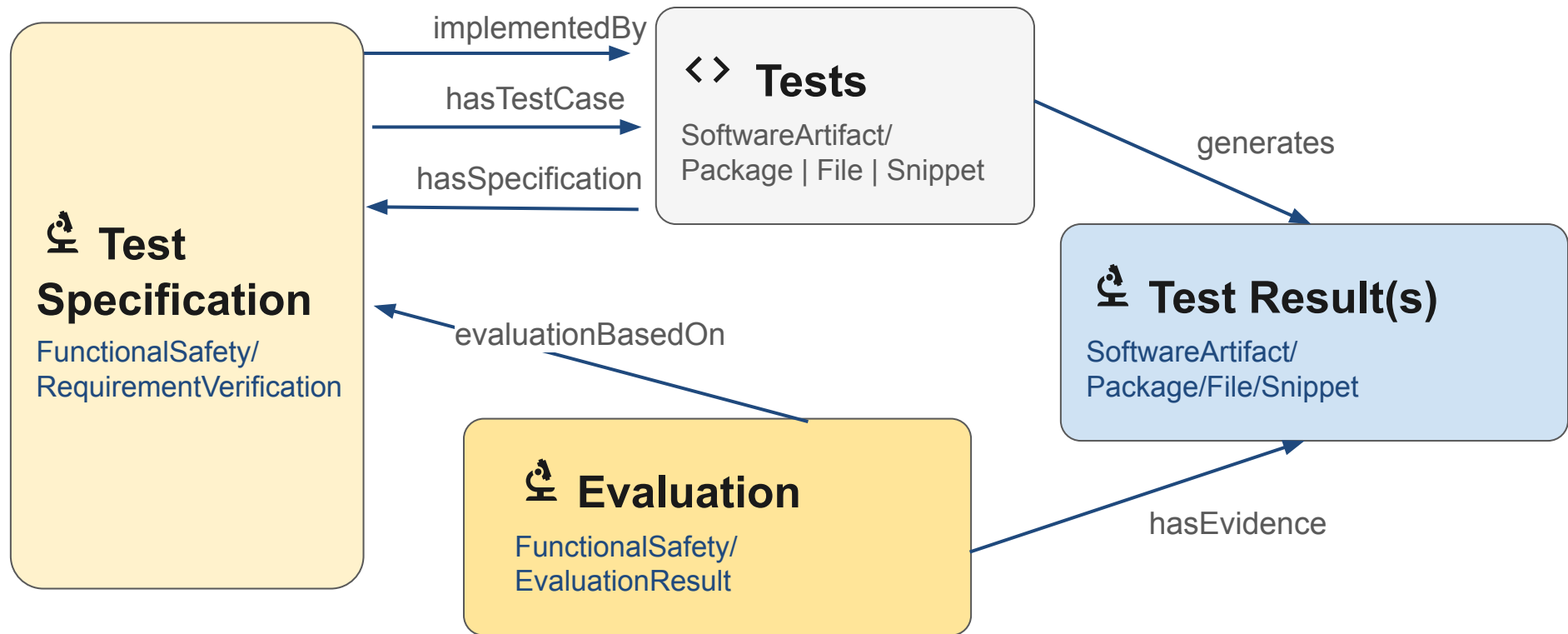
Looking to add to relationship as requirements can be reused, evolved, etc. but once created not modified.

RequirementVerification (active development)

Field Purpose	Cardinality	SPDX 3.1
Identifier of RequirementVerification	(1,1)	core/ spdxID FunctionalSafety/ verificationUID
VerificationMethod	(1,1)	FunctionalSafety/ VerificationType
Why of method selected	(0,1)	core/ Rationale
Any conditions to satisfy after verification performed	(0,1)	FunctionalSafety/ VerificationPostcondition
Any conditions to be satisfied before verification is performed	(0,1)	FunctionalSafety/ VerificationPrecondition

VerificationType is one of **analysis** | **assessment** | **audit** | **demonstration** | **inspection** | **review** | **test** | **other**

SPDX 3.1-dev (active development)



EvidenceRelationship (active development)

hasEvidence: Every **to** Element is considered as evidence for the **from** Element (**from** hasEvidence **to**).

Field Purpose	Cardinality	SPDX 3.1
Evidence identifier	(1,1)	core/ spdxID FunctionalSafety/ EvidenceUID
Enumeration of types of Evidence Artifacts	(0,*)	FunctionalSafety/ EvidenceType
from	(1,1)	Relationship from RequirementVerification
to	(1,*)	Relationship to objects that provide evidence

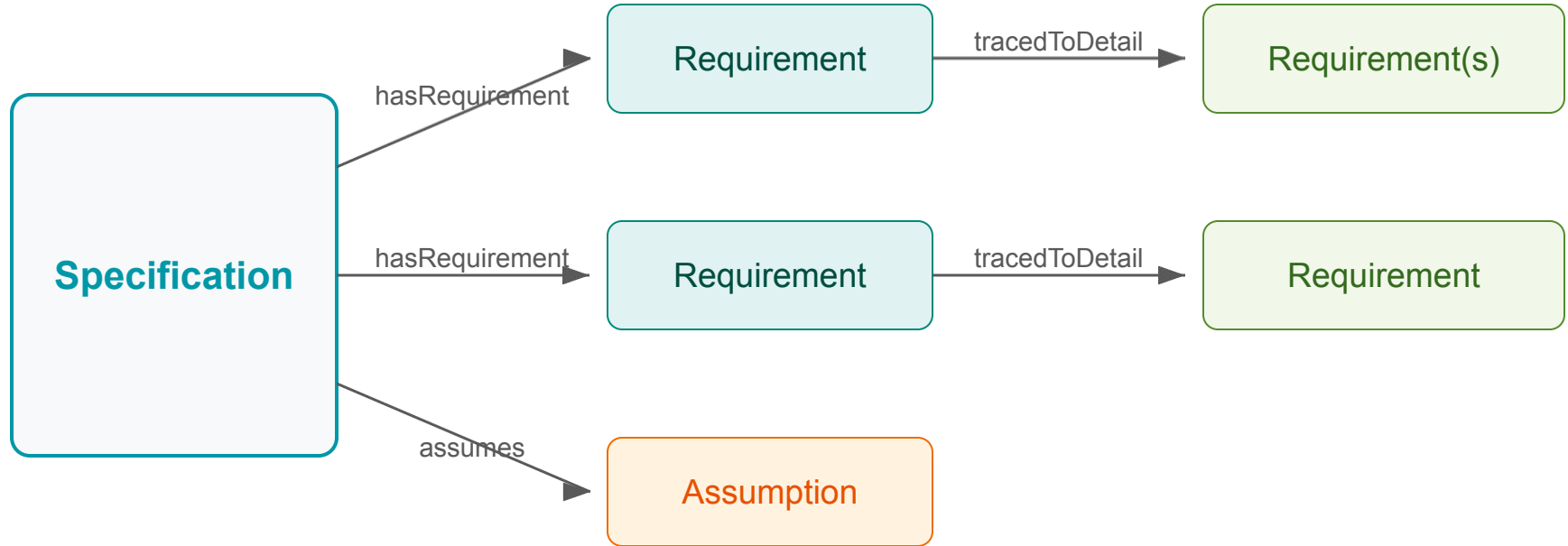
EvidenceType is one of **log** | **observation** | **recording** | **report** | **other**

Evaluation Result (active development)

Field Purpose	Cardinality	SPDX 3.1
Evaluation Identifier	(1,1)	core/ spdxID
Result of Evaluation	(1,1)	FunctionalSafety/EvaluationResult/ evaluation which is an evaluationResultType
Rationale	(1,1)	core/element
Evaluation based on	(1,1)	FunctionalSafety/EvaluationResult/evaluationBasedOn which is a RequirementVerification

evaluationResultType is one of **pass** | **fail** | **inconclusive**

SPDX 3.1-dev (active development)



Specification (active development)

Field Purpose	Cardinality	SPDX 3.1
Identify a Specification Artifact	(1,1)	core/ spdxID core/ externalIdentifier
Type of Specification	(0,1)	core/ specType
Name of specification	(0,1)	xsd:string

specificationType is one of **formalStandard** | **regulation** | **specification** | **other**

Assumption of use (active development)

Field Purpose	Cardinality	SPDX 3.1
Identifier of Assumption	(1,1)	core/ spdxID FunctionalSafety/ assumptionUID
Assumption of use Statement	(1,1)	FunctionalSafety/ assumptionStatement
why of assumption	(0,1)	core/ Rationale
Assumption Integrity	(1,1)	FunctionalSafety/ verifiedUsing (see Integrity Method for options)

Note: assumptions should be able to point to other assumptions via a **"assumes"** relationship.

Next Steps

- 1) Complete the SPDX formalism definition to maintain requirements, evidence, test specifications along the corresponding code and test cases
- 2) Develop tools (e.g. BASIL) to support such a formalism and relationships
- 3) Define a change management process leveraging tools and formalism from points above
- 4) Create a repo for requirements and test specifications and start populating it with the Linux kernel data; and work with interested kernel maintainers and KernelCI community to review.
- 5) Spot bugs in existing code and in new contributions, show the value and create an ecosystem of interested parties using all of the above

Join us!

[Safety-Architecture WG](#)



ELISA
Enabling **Linux** in
Safety Applications

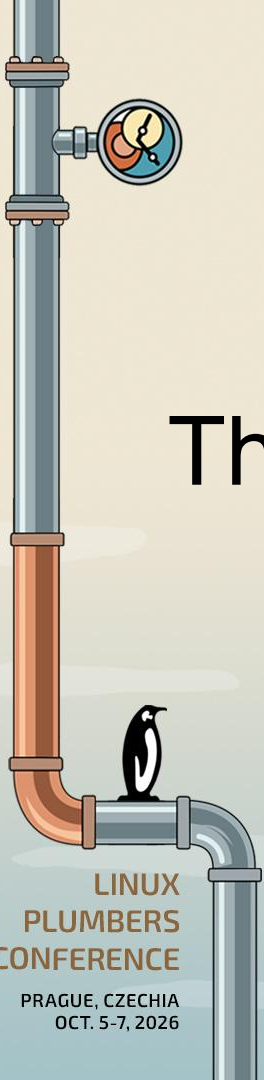
WG Regular Public Meeting

We meet every Friday 4-5pm CEST

All ELISA public meetings can be accessed here <https://zoom-lfx.platform.linuxfoundation.org/meetings/elisa>

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

A vertical pipe with a pressure gauge and a penguin on a horizontal section. The pipe is grey with orange-brown joints. A circular pressure gauge with a yellow face and blue needle is attached to the upper part of the vertical pipe. At the bottom, the pipe turns 90 degrees into a horizontal section where a small black and white penguin is perched. The background is a light beige gradient with faint, horizontal, light blue wavy lines.

Thank you

LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA
OCT. 5-7, 2026

Licensing of LPC Results

All work created during the LPC 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.