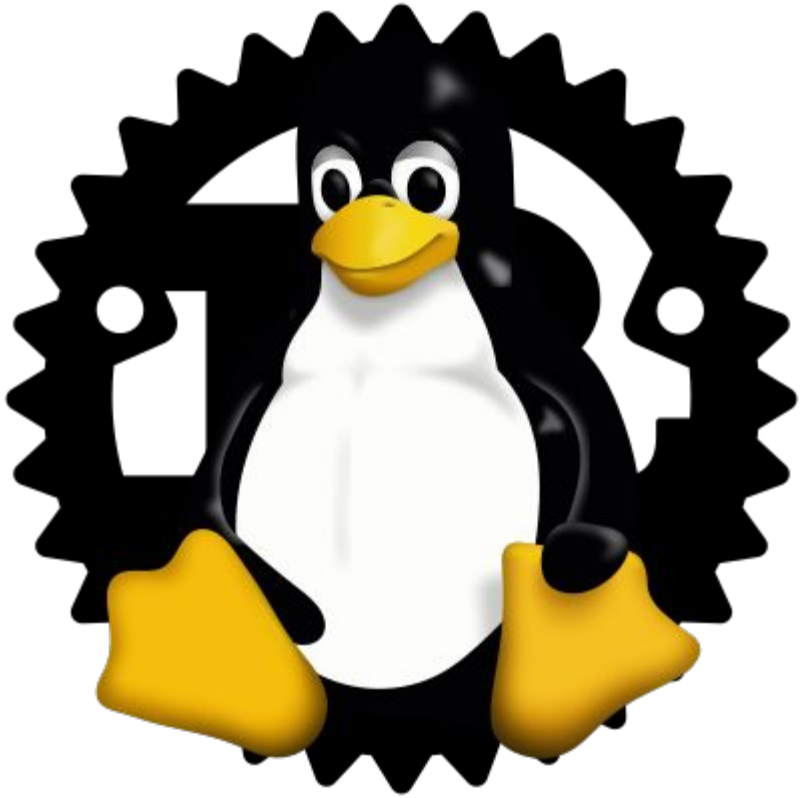


Testing in-kernel Rust code

Miguel Ojeda

ojeda@kernel.org

Rust built-in testing features

```rust
/// Returns `x + 1`.
pub fn function_with_a_bug(x: i32) -> i32 {
    x + 9
}

/// Panics if `x < 0`.
pub fn function_that_may_panic(x: i32) {
   if x < 0 {
       panic!("x must be positive");
   }
}

/// Returns `x + 1`, but fails if `x < 0`.
pub fn function_that_may_fail(x: i32) -> Result<i32, ()> {
   if x < 0 {
       return Err(());
   }

   Ok(x + 1)
}
```

```rust
#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn test_that_succeeds() {
        assert_eq!(2 + 2, 4);
    }

    #[test]
    fn test_that_fails() {
        assert_eq!(43, function_with_a_bug(42));
    }
}
```


```rust
#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn test_that_succeeds() {
        assert_eq!(2 + 2, 4);
    }

    #[test]
    fn test_that_fails() {
        assert_eq!(43, function_with_a_bug(42));
    }
}
```

(or not, i.e. unit tests vs. integration tests)

```rust
#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn test_that_succeeds() {
        assert_eq!(2 + 2, 4);
    }

    #[test]
    fn test_that_fails() {
        assert_eq!(43, function_with_a_bug(42));
    }
}
```

```rust
#[cfg(test)]
mod tests {
    use super::*;

    // ...

    #[test]
    fn test_that_panics() {
        panic!("oops");
    }

    #[test]
    #[should_panic]
    fn test_that_is_expected_to_panic() {
        function_that_may_panic(-42);
    }
}
```

```rust
#[cfg(test)]
mod tests {
    use super::*;

    // ...

    #[test]
    #[ignore]
    fn test_that_is_ignored() {
        // Code that takes an hour to run
    }
}
```

```rust
#[cfg(test)]
mod tests {
    use super::*;

    // ...

    #[test]
    fn test_based_on_result() -> Result<(), ()> {
        let x = function_that_may_fail(42)?;
        function_that_may_fail(x)?;
        Ok(())
    }
}
```

```rust
#[cfg(test)]
mod tests {
    use super::*;

    // ...

    use test::Bencher;

    #[bench]
    fn benchmark(b: &mut Bencher) {
        b.iter(|| function_with_a_bug(42));
    }
}
```

```rust
#[cfg(test)]
mod tests {
    use super::*;

    // ...

    use test::Bencher;

    #[bench]
    fn benchmark(b: &mut Bencher) {
        b.iter(|| function_with_a_bug(42));
    }
}
```

(unstable feature)

```
$ cargo test
    Finished test [unoptimized + debuginfo] target(s) in 0.59s
     Running unittests (target/debug/deps/example-26c4ff7654c615cf)

running 7 tests
test tests::benchmark ... ok
test tests::test_based_on_result ... ok
test tests::test_that_fails ... FAILED
test tests::test_that_is_expected_to_panic - should panic ... ok
test tests::test_that_is_ignored ... ignored
test tests::test_that_panics ... FAILED
test tests::test_that_succeeds ... ok

failures:

---- tests::test_that_fails stdout ----
thread 'tests::test_that_fails' panicked at 'assertion failed: `(left == right)`
  left: `43`,
 right: `51`', src/lib.rs:66:9
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace

---- tests::test_that_panics stdout ----
thread 'tests::test_that_panics' panicked at 'oops', src/lib.rs:73:9


failures:
    tests::test_that_fails
    tests::test_that_panics

test result: FAILED. 4 passed; 2 failed; 1 ignored; 0 measured; 0 filtered out; finished in 0.00s

error: test failed, to rerun pass '--lib'
```

```rust
/// Returns `x + 1`.
///
/// # Examples
///
/// ```
/// assert_eq!(43, example::function_with_a_bug(42));
/// ```
pub fn function_with_a_bug(x: i32) -> i32 {
    x + 9
}
```

```rust
/// Returns `x + 1`.
///
/// # Examples
///
/// ```
/// assert_eq!(43, example::function_with_a_bug(42));
/// ```
///
/// ```ignore
/// // Code that takes an hour to run
/// ```
///
/// ```no_run
/// let x = "only needs to compile";
/// ```
///
/// ```compile_fail
/// let x = "fails to" 42 "compile";
/// ```
pub fn function_with_a_bug(x: i32) -> i32 {
    x + 9
}
```

```rust
/// Panics if `x < 0`.
///
/// # Examples
///
/// ```
/// # use example::*;
/// function_that_may_panic(42); // no panic
/// ```
///
/// ```should_panic
/// # use example::*;
/// function_that_may_panic(-42); // panics!
/// ```
pub fn function_that_may_panic(x: i32) {
    if x < 0 {
        panic!("x must be positive");
    }
}
```

```
...

   Doc-tests example

running 6 tests
test src/lib.rs - function_with_a_bug (line 12) ... ignored
test src/lib.rs - function_with_a_bug (line 16) - compile ... ok
test src/lib.rs - function_with_a_bug (line 20) - compile fail ... ok
test src/lib.rs - function_that_may_panic (line 31) ... ok
test src/lib.rs - function_that_may_panic (line 36) ... ok
test src/lib.rs - function_with_a_bug (line 8) ... FAILED

failures:

---- src/lib.rs - function_with_a_bug (line 8) stdout ----
Test executable failed (exit code 101).

stderr:
thread 'main' panicked at 'assertion failed: `(left == right)`
  left: `43`,
 right: `51`', src/lib.rs:4:1
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace



failures:
    src/lib.rs - function_with_a_bug (line 8)

test result: FAILED. 4 passed; 1 failed; 1 ignored; 0 measured; 0 filtered out; finished in 0.48s

error: test failed, to rerun pass '--doc'
```

```
$ cargo bench
    Finished bench [optimized] target(s) in 0.01s
     Running unittests (target/release/deps/example-3d9ffbf4b0a92572)

running 7 tests
test tests::test_based_on_result ... ignored
test tests::test_that_fails ... ignored
test tests::test_that_is_expected_to_panic - should panic ... ignored
test tests::test_that_is_ignored ... ignored
test tests::test_that_panics ... ignored
test tests::test_that_succeeds ... ignored
test tests::benchmark ... bench:           0 ns/iter (+/- 0)

test result: ok. 0 passed; 0 failed; 6 ignored; 1 measured; 0 filtered out; finished in 0.72s
```

Doctests in the documentation

Crate example

Version 0.1.0

See all example's items

Functions

Crates

example

Crate example [-][src]

Functions

function_that_may_fail	Returns `x + 1`, but fails if `x < 0`.
function_that_may_panic	Panics if `x < 0`.
function_with_a_bug	Returns `x + 1`.

Function example::function_that_may_panic [-][src]

```
pub fn function_that_may_panic(x: i32)
```

[-] Panics if `x < 0`.

Examples

```
function_that_may_panic(42); // no panic
```

```
function_that_may_panic(-42); // panics!
```

Other items in **example**

Functions

function_that_may_fail
function_that_may_panic
function_with_a_bug

Other items in **example**

Functions

function_that_may_fail
function_that_may_panic
function_with_a_bug

Function example::function_with_a_bug [-][src]

```
pub fn function_with_a_bug(x: i32) -> i32
```

[-] Returns `x + 1`.

Examples

```
assert_eq!(43, example::function_with_a_bug(42));
```

```
// Code that takes an hour to run
```

```
let x = "only needs to compile";
```

```
let x = "fails to" 42 "compile";
```

All crates | Click or press 'S' to search, '?' for more options...

Other items in **example**

Functions

function_that_may_fail
function_that_may_panic
function_with_a_bug

Function example::function_with_a_bug [-][src]

```
pub fn function_with_a_bug(x: i32) -> i32
```

[-] Returns `x + 1`.

Examples

```
assert_eq!(43, example::function_with_a_bug(42));
```

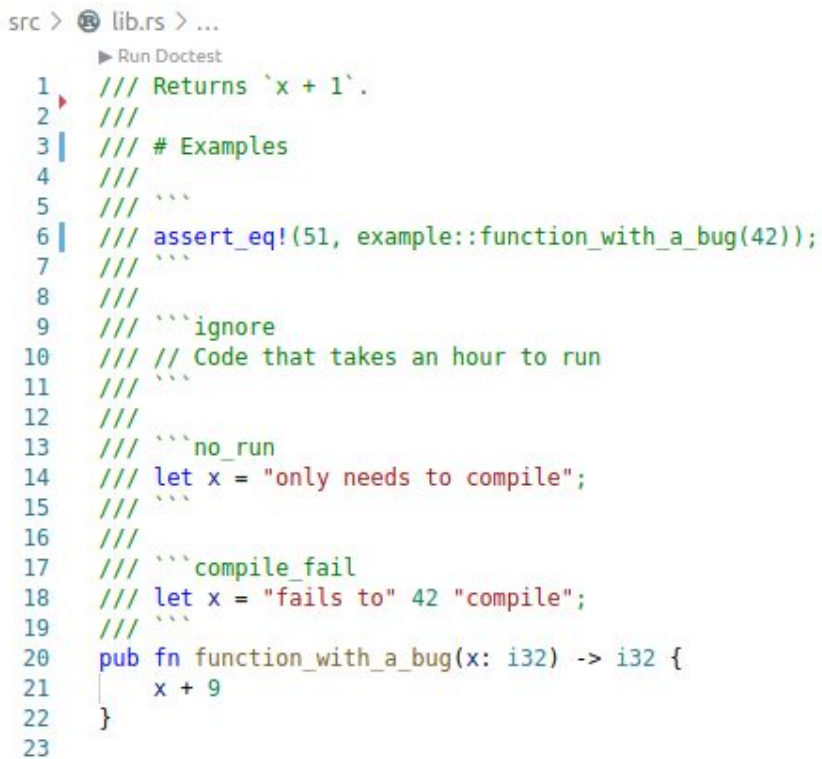
```
// Code that takes an hour to run
```

```
let x = "only needs to compile";
```

```
let x = "fails to" 42 "compile";
```

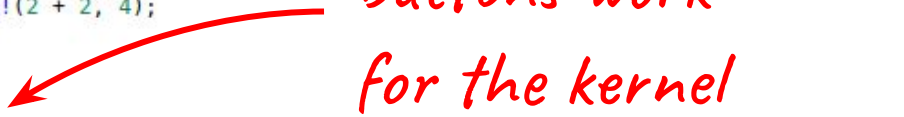
Doc generator is aware of the type of test

rust-analyzer IDE support


src > lib.rs > ...
▶ Run Doctest
/// Returns `x + 1`.
///
/// # Examples
///
/// ```
/// assert_eq!(51, example::function_with_a_bug(42));
/// ```
///
/// ```ignore
/// // Code that takes an hour to run
/// ```
///
/// ```no_run
/// let x = "only needs to compile";
/// ```
///
/// ```compile_fail
/// let x = "fails to" 42 "compile";
/// ```
pub fn function_with_a_bug(x: i32) -> i32 {
x + 9
}

```
52  #[cfg(test)]
    ▶ Run Tests | Debug
53  mod tests {
54      use super::*;
55
56      #[test]
        ▶ Run Test | Debug
57      fn test_that_succeeds() {
58          assert_eq!(2 + 2, 4);
59      }
60
61      #[test]
        ▶ Run Test | Debug
62      fn test_that_fails() {
63          assert_eq!(51, function_with_a_bug(42));
64      }
65
66      #[test]
        ▶ Run Test | Debug
67      fn test_that_panics() {
68          //panic!("oops");
69      }
70
71      #[test]
72      #[should_panic]
        ▶ Run Test | Debug
73      fn test_that_is_expected_to_panic() {
74          function_that_may_panic(-42);
75      }
```

```
52  #[cfg(test)]
    ▶ Run Tests | Debug
53  mod tests {
54      use super::*;
55
56      #[test]
        ▶ Run Test | Debug
57      fn test_that_succeeds() {
58          assert_eq!(2 + 2, 4);
59      }
60
61      #[test]
        ▶ Run Test | Debug
62      fn test_that_fails() {
63          assert_eq!(51, function_with_a_bug(42));
64      }
65
66      #[test]
        ▶ Run Test | Debug
67      fn test_that_panics() {
68          //panic!("oops");
69      }
70
71      #[test]
72      #[should_panic]
        ▶ Run Test | Debug
73      fn test_that_is_expected_to_panic() {
74          function_that_may_panic(-42);
75      }
```



In the kernel

```rust
fn trim_whitespace(mut data: &[u8]) -> &[u8] {
    // ...
}

#[cfg(test)]
mod tests {
   use super::*;

   #[test]
   fn test_trim_whitespace() {
       assert_eq!(trim_whitespace(b"foo    "), b"foo");
       assert_eq!(trim_whitespace(b"    foo"), b"foo");
       assert_eq!(trim_whitespace(b"  foo  "), b"foo");
   }
}
```

```rust
/// Wraps the kernel's `struct task_struct`.
///
/// # Invariants
///
/// The pointer `Task::ptr` is non-null and valid. Its reference count is also non-zero.
///
/// # Examples
///
/// The following is an example of getting the PID of the current thread with
/// zero additional cost when compared to the C version:
///
/// ```
/// # use kernel::prelude::*;
/// use kernel::task::Task;
///
/// # fn test() {
/// Task::current().pid();
/// # }
/// ```
pub struct Task {
    pub(crate) ptr: *mut bindings::task_struct,
}
```

```rust
/// Wraps the kernel's `struct task_struct`.
///
/// # Invariants
///
/// The pointer `Task::ptr` is non-null and valid. Its reference count is also non-zero.
///
/// # Examples
///
/// The following is an example of getting the PID of the current thread with
/// zero additional cost when compared to the C version:
///
/// ```
/// # use kernel::prelude::*;
/// use kernel::task::Task;
///
/// # fn test() {
/// Task::current().pid();
/// # }
/// ```
pub struct Task {
    pub(crate) ptr: *mut bindings::task_struct,
}
```

Doc tests already caught a mistake

```
/// Getting the current task and storing it in some struct. The reference count is automatically
/// incremented when creating `State` and decremented when it is dropped:
///
/// ```
/// # use kernel::prelude::*;
/// use kernel::task::Task;
///
/// struct State {
///     creator: Task,
///     index: u32,
/// }
///
/// impl State {
///     fn new() -> Self {
///         Self {
///             creator: Task::current().clone(),
///             index: 0,
///         }
///     }
/// }
/// ```
```

Ideas

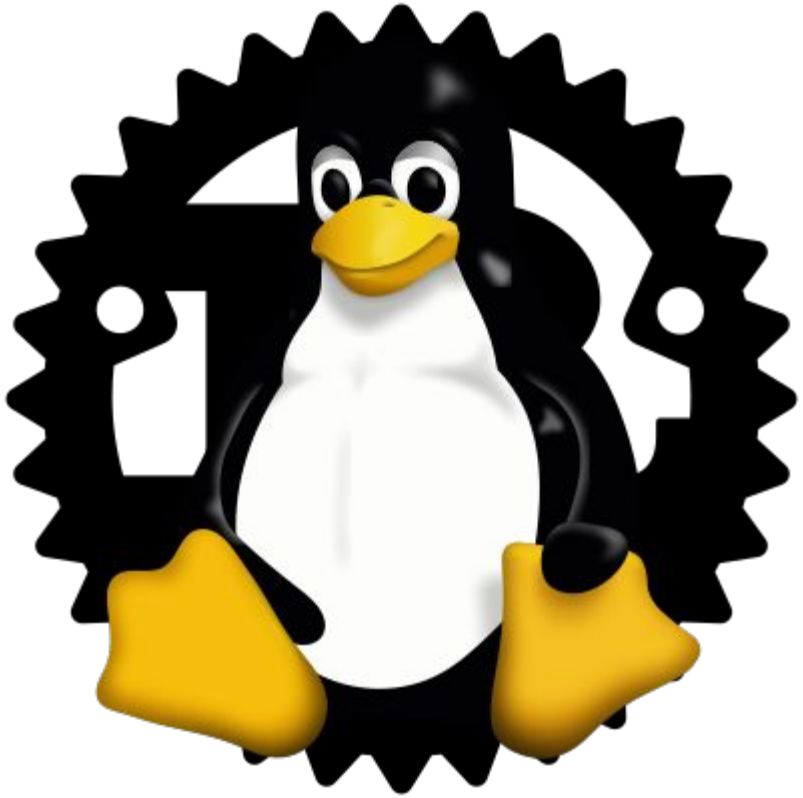
```rust
#[test]
#[host]
fn test_that_runs_in_the_host() {
    // Something that can be tested in the host.
}

#[test]
#[user]
fn test_that_runs_in_the_target's_userspace() {
    // Something that must be tested in the target,
    // but the test runs in userspace.
}

#[test]
#[kernel]
fn test_that_runs_in_the_target's_kernelspace() {
    // Something that must be tested in the target,
    // but the test runs in kernelspace.
}
```

```
/// ```host
/// assert_eq!(2 + 2, 4);
/// ```
///
/// ```user
/// assert_eq!(2 + 2, 4);
/// ```
///
/// ```kernel
/// assert_eq!(2 + 2, 4);
/// ```
pub fn f() {
    // ...
}
```

Discussion



Testing in-kernel Rust code

Miguel Ojeda

ojeda@kernel.org