**Creating an OS using Rust: [Part-4] Println! Macro Implementation**

[[](https://tdtapas.medium.com/?source=post_page-----f17da1d5fbdd--------------------------------)](https://tdtapas.medium.com/?source=post_page-----f17da1d5fbdd--------------------------------)

[Tapas Das](https://tdtapas.medium.com/?source=post_page-----f17da1d5fbdd--------------------------------)

·

Follow

3 min read

·

16 hours ago

1

1

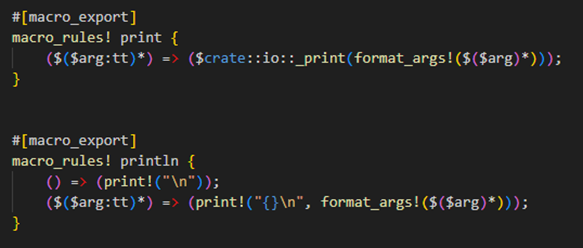


In [Part-3](https://medium.com/dev-genius/creating-an-os-using-rust-part-3-playing-with-vga-text-mode-c0a7120ac265) of this blog series, we had discussed about writing a safe interface for accessing and writing to VGA text buffer, so as to avoid using the Rust unsafe block. In this blog post, we will build on same and recreate the *println!* and *panic!* macros to ease up the process of writing characters or strings on boot up.

Let’s get started.

**Macro syntax for println**

We will start by looking up the macro syntax for *println* in the standard library.



The *println* macro has two rules:

1. The first rule is for invocations without arguments, e.g., println!(), which is expanded to **print!(“\n”)** and thus just prints a newline.
2. The second rule is for invocations with parameters such as **println!(“Hello”)** or **println!(“Number: {}”, 100)**. It is also expanded to an invocation of the *print! macro*, passing all arguments and an additional newline \n at the end.

The *print!*macro expands to a call of the *\_print function* in the io module. The *$crate variable* ensures that the macro also works from outside the std crate by expanding to std when it’s used in other crates.

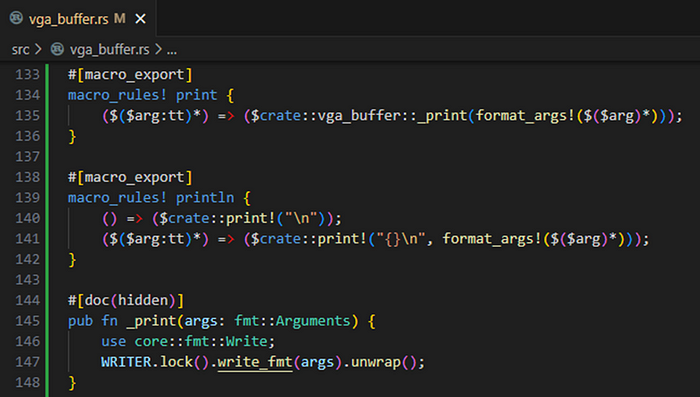
For more information, visit below link.

**[println in std - Rust](https://doc.rust-lang.org/nightly/std/macro.println.html?source=post_page-----f17da1d5fbdd--------------------------------" \t "_blank)**

[Prints to the standard output, with a newline.](https://doc.rust-lang.org/nightly/std/macro.println.html?source=post_page-----f17da1d5fbdd--------------------------------" \t "_blank)

[doc.rust-lang.org](https://doc.rust-lang.org/nightly/std/macro.println.html?source=post_page-----f17da1d5fbdd--------------------------------" \t "_blank)

We will use these macro definitions and modify them to use our own *\_print* function.



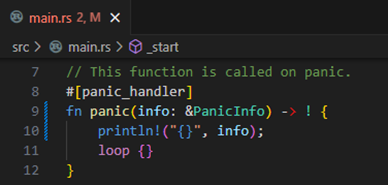
The \_print function locks our static *WRITER* and calls the *write\_fmt* method on it. This method is from the Write trait, which we need to import.

Also, same as the standard library, we have added the *#[macro\_export] attribute* to both macros to make them available everywhere in our crate. This will place the macros in the root namespace of the crate, so importing them via use **crate::vga\_buffer::println** does not work. Instead, we have to use **crate::println**.

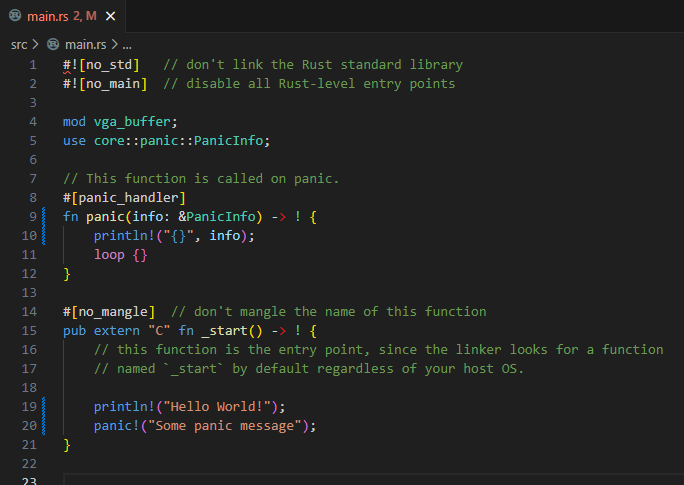
*Note: The*#[doc(hidden)]*attribute hides the private implementation of \_print function from generated documentation.*

**Printing panic messages**

Now that we have got our very own *println macro*, we can use it in our panic function to print the panic message and the location of the panic.

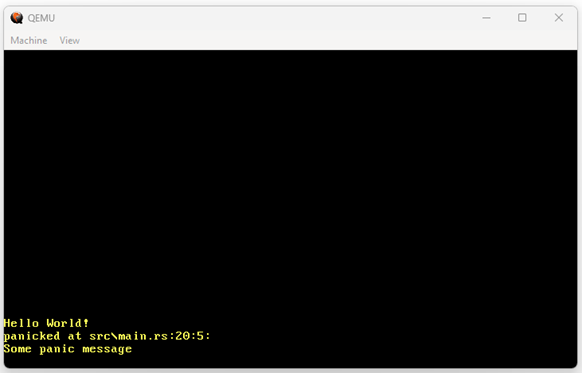


Below is the final main.rs script with modified changes for both *println* and *panic* macros.



*Note that we don’t have to import the macro in the main function, because it already lives in the root namespace.*

Let’s build the bootimage once more and print to screen using both *println* and *panic* macros.



Et Voila! As expected, we now see a *“Hello World!”* on the screen. Also notice that we know not only that a panic has occurred, but also the panic message and where in the code it happened.

**Summary**

Finally, we have got our very own **println** and **panic** macros that can be used from anywhere in the codebase without going through the hassle of importing them.

Also, we are at a stage now where we need to think about testing too and how to set up Rust’s built-in unit test framework. This will help us in writing basic unit tests for the VGA buffer module.

I’ll cover the detailed implementation steps in the next post.

[Rust](https://medium.com/tag/rust?source=post_page-----f17da1d5fbdd---------------rust-----------------)

[Operating Systems](https://medium.com/tag/operating-systems?source=post_page-----f17da1d5fbdd---------------operating_systems-----------------)

[Macro](https://medium.com/tag/macro?source=post_page-----f17da1d5fbdd---------------macro-----------------)