Async with Example

Rusty weirdness

Rust Async Programming

- **async** in Rust may be a bit different from what you're used to in other languages. Having done asynchronous coding mostly in JavaScript and C#, it certainly was to me. Here's a few key things to understand:

Rust Async Programming

- An **async** function does not (necessarily) start executing immediately
- To start an **asynchronous** function, you must either .await it or launch a task using an executor (we'll get to that in a moment). Until this happens, all you have is a **Future** that has not started. Let's look at an example to make it clearer:

Rust Async Programming

```
use async_std::task;
async fn negate_async(n: i32) -> i32 {
    println!("Negating {}", n);
    task::sleep(std::time::Duration::from_secs(5)).await;
    println!("Finished sleeping for {}!", n);
    n * -1
}
```

Rust Async Programming

```
async fn f() -> i32 {
    let neg = negate_async(1);
    let neg_task = task::spawn(negate_async(2));
    task::sleep(std::time::Duration::from_secs(1)).await;
    neg.await + neg_task.await
}
```

```
[dependencies]
async-std = "1"
```

Rust Async Programming

- The first line imports **async_std::task**. There's more on this below, but we need an external library to run futures as the standard library does not come with an **executor**.
- The async function **negate_async** takes as input a signed integer, sleeps for 5 seconds, and returns the negated version of that integer.

Rust Async Programming

- The async function f is more interesting:
- The first line (let neg ...) creates a Future of the negate_async function and assigns it to the neg variable. Importantly, it does not start executing yet.
- The next line of code (let neg_task ...) uses the task::spawn function to start executing the Future returned by negate_async. Like with neg, the Future returned by negate_async is assigned to the neg_task variable.

Rust Async Programming

- Next: we sleep for a second. This is so that it will be obvious from the output when a task starts running.
- Finally, we await both futures, add them together, and return them. By awaiting neg, we start executing the Future and run it to completion. Since neg_task has already been started, we just wait for it to finish.

Resources

Book : https://rust-lang.github.io/async-book/

Link to the article : https://thomashartmann.dev/blog/async-rust/

Source code repository : https://github.com/PIAIC-IOT/Quarter3-Online.git

Summary