perplexity

Tasks spawned by tokio::spawn must implement Send. This allows the Tokio runtime to move the tasks between threads while they are suspended at an .await.

give me a example

To demonstrate how tasks spawned by `tokio::spawn` must implement `Send`, here's a simple example in Rust using the Tokio runtime. This example illustrates how to create an asynchronous task that safely shares data between threads.

Example Code

```
use tokio::task;
use std::sync::{Arc, Mutex};

#[tokio::main]
async fn main() {
    // Create a shared counter wrapped in Arc and Mutex
    let counter = Arc::new(Mutex::new(0));

    // Clone the Arc to share ownership with the spawned task
    let counter_clone = Arc::clone(&counter);

    // Spawn a new task
    let handle = tokio::spawn(async move {
        // Lock the mutex to safely access the counter
        let mut num = counter_clone.lock().unwrap();
        *num += 1; // Increment the shared counter
        println!("Counter in task: {}", num);
    });

    // Await the completion of the spawned task
    handle.await.unwrap();

    // Lock the mutex again to read the final value of the counter
    let final_count = counter.lock().unwrap();
    println!("Final Counter: {}", final_count);
}
```

Explanation

- **Arc and Mutex**: The `counter` is wrapped in an `Arc` (Atomic Reference Counted) and `Mutex` to allow safe concurrent access from multiple tasks. The `Arc` provides shared ownership, while the `Mutex` ensures that only one thread can access the data at a time.
- **Spawning a Task**: The `tokio::spawn` function is used to create a new asynchronous task. The closure passed to `spawn` must implement `Send`, which is guaranteed here because both `Arc` and `Mutex` are `Send`.
- **Using `.await`**: The main function awaits the completion of the spawned task using `handle.await.unwrap()`. This ensures that the main function does not exit before the spawned task completes.

This example illustrates how to use Tokio's asynchronous capabilities while adhering to Rust's safety guarantees regarding thread-safety and data sharing across tasks.

⁂