```rust
use std::collections::VecDeque;
use std::sync::{Arc, Mutex};

// Task Trait
trait Task {
    fn execute(&self) -> usize;
}

// SumTask struct
struct SumTask {
    n1: usize,
    n2: usize,
}

impl SumTask {
    fn new(n1: usize, n2: usize) -> Self {
        Self { n1, n2 }
    }
}

impl Task for SumTask {
    fn execute(&self) -> usize {
        self.n1 + self.n2
    }
}

// LenTask struct
struct LenTask {
    s: String,
}

impl LenTask {
    fn new(s: String) -> Self {
        Self { s }
    }
}


```

```rust
impl Task for LenTask {
    fn execute(&self) -> usize {
        self.s.len()
    }
}

// Shared task queue type
type TaskQueue = Arc<Mutex<VecDeque<Box<dyn Task>>>>;

// Tasker struct
struct Tasker {
    queue: TaskQueue,
}

impl Tasker {
    // Create a new Tasker
    fn new() -> Self {
        Self {
            queue: Arc::new(Mutex::new(VecDeque::new())),
        }
    }

    // Schedule a task
    fn schedule_task(&mut self, task: Box<dyn Task>) {
        let mut queue = self.queue.lock().unwrap();
        queue.push_back(task);
    }

    // Create a new Tasker linked to the current queue
    fn get_tasker(&self) -> Tasker {
        Tasker {
            queue: Arc::clone(&self.queue),
        }
    }

    // Create an Executer linked to the current queue
    fn get_executer(&self) -> Executer {
        Executer {
            queue: Arc::clone(&self.queue),
        }
    }
```

```
}

// Executer struct
struct Executer {
    queue: TaskQueue,
}

impl Executer {
    // Execute the next task in the queue
    fn execute_task(&mut self) -> Option<usize> {
        let mut queue = self.queue.lock().unwrap();
        queue.pop_front().map(|task| task.execute())
    }
}
```
