Day 24

The 64-Core Sleigh Hustle

The Story

"Alright, listen up! The sleigh's armed with the ElfiX 9000—64 blazing-fast cores. But without a working task queue, we're going nowhere. I need ideas, and I need them fast!"

Blitzen stretched lazily, balancing a candy cane on his nose. "What's the big deal? Just make a list, throw everything on there, and let the sleigh do its thing."

The Story

Bernard, the Head Elf, sighed audibly. "Blitzen, a list? Really? We need something efficient, predictable, and thread-safe. We're not juggling reindeer treats here!"

Santa grunted. "Bernard's right. This isn't reindeer games. We need ultimate thread safety and dynamic dispatch. Bernard, go on."

The Story

Bernard adjusted his glasses. "We use a `VecDeque`. Tasks get pushed to the back and popped from the front. It's simple and linear."

Blitzen tilted his head. "And how do we stop the sleigh from pulling a Rudolph and wandering off the rails?"

The Story

Bernard rolled his eyes. "Thread safety. Wrap the `VecDeque` in a `Mutex`, so each core pops tasks one at a time without stepping on each other's hooves."

Santa stroked his beard. "And the tasks? We've got a mix: deliveries, routing, even…" he checked the list, "…this goat thing."

The Story

Bernard cut him off. "We implement a `SleighTask` trait. Each task adheres to it, and the queue just holds `Box<dyn SleighTask>`. The cores process whatever's next without worrying about specifics."

Blitzen yawned. "Sounds overly complicated to me."

"Spoken like someone who's never touched a `Mutex`," Bernard snapped.

The Story

Santa chuckled. "Enough yapping! The `VecDeque` is our solution. Let's build it. Christmas flies on this queue!"

The workshop buzzed with determination. With the `VecDeque` at its heart, the sleigh would soon conquer the skies—no reindeer complaints included.

Your Mission

- The `SantaSleighQueue` should have a field `records`, make sure it is thread safe and can be mutated by multiple threads.
- The `records` list should accept either of the two task types: `ElfTask` and `ReindeerTask`
- Both task types should implement the `SleighTask` trait

Your Mission

- The `SantaSleighQueue` should have these methods:
  - `new() -> Self` : Creates a new `SantaSleighQueue`
  - `enqueue` adds a task to the back of the queue, returns `()`
  - `get_task` pops the next task from the front of the queue, returns `Option<T>`

Your Mission

- The `ElfTask` should have these fields:
  - `name: String`
  - `urgency: u32`
- The `ElfTask` should have a `new()` associated function that creates a new `ElfTask`
- The `ReindeerTask` should have these fields:
  - `name: String`
  - `weight: String`

Your Mission

- The `ReindeerTask` should have a `new()` associated function that creates a new `ReindeerTask`
- Don't worry about the use of the `urgency` and `weight` fields, they are just there for demonstration purposes
- You can use `unwrap()` on the mutex lock result; for this challenge, you don't need to worry about poisoning.

Your Mission

- Make sure all important values are `pub` so they can be accessed from outside the module
- Make sure you have a look at the bottom of the code to see how Santa wants to use the `SantaSleighQueue` API.

Hints

If you're unsure where to start, take a look at these tips:

- Use a `VecDeque` to store the tasks, so that you can push and pop tasks from the front and back.
- Wrap the `VecDeque` in a `Mutex` to make it thread-safe.

Hints

- In order to hold either `ElfTask` or `ReindeerTask` in the `VecDeque`, you can use `Box` to store a trait object on the heap.

```rust
pub struct SantaSleighQueue {
    record: Mutex<VecDeque<Box<dyn SleighTask>>>,
}
```

Hints

- Before mutating the `VecDeque`, you need to lock the `Mutex` to ensure that only one thread can access it at a time.

```rust
let mut records = self.records.lock().unwrap();
```

Hints

- Since `Mutex` allows interior mutability, you don't need to get a mutable reference to the `self` object, you can just use `&self`.

```
pub fn enqueue(&self, task: Box<dyn SleighTask>) {
    self.record.lock().unwrap().push_back(task);
}
```