### **Lab 6:** **Concurrency in Rust**

**Exercise: Concurrent Sum Calculation**

In this exercise, we will create a Rust program that calculates the sum of numbers using multiple threads concurrently. The program will split the task of summing numbers among different threads and then combine the results to get the final sum.

1. Create a new Rust project using cargo:
2. Open your terminal/command prompt and run the following command:

cargo new concurrent\_sum

cd concurrent\_sum

1. Open the main.rs file in the src directory of your project. You can use any code editor for this.
2. Import the necessary modules for working with threads:

use std::thread;

1. Implement a function called sum\_range that calculates the sum of numbers within a given range:

fn sum\_range(start: u32, end: u32) -> u32 {

let mut sum = 0;

for num in start..=end {

sum += num;

}

sum

}

In the main function, create two threads to calculate the sum concurrently for two different ranges:

fn main() {

const N: u32 = 100\_000;

const THREADS: usize = 2;

let mut handles = vec![];

for i in 0..THREADS {

let start = i \* (N / THREADS as u32) + 1;

let end = (i + 1) \* (N / THREADS as u32);

let handle = thread::spawn(move || sum\_range(start, end));

handles.push(handle);

}

let mut final\_sum = 0;

for handle in handles {

final\_sum += handle.join().unwrap();

}

println!("Sum of numbers from 1 to {}: {}", N, final\_sum);

}

Save the file and return to your terminal/command prompt.

Build and run your program using cargo run:

cargo run

The program will calculate the sum of numbers from 1 to 100,000 using multiple threads concurrently and display the final result.

Example Output:

Sum of numbers from 1 to 100000: 5000050000

We have now successfully completed the lab exercise on concurrency in Rust programming! We learned how to utilize multiple threads to perform tasks concurrently and combine the results to get the final outcome. Concurrency is a powerful feature in Rust, allowing you to efficiently utilize modern multi-core processors and improve performance for certain tasks.

**Happy coding!**